CAB420 – Assignment 2

Phuoc Nguyen – N9951733
Long Thai – N10006257

Part A: SVMs and Bayes Classifiers

Support Vector Machines

1. For each dataset, provide a rationale for which kernel should be the best for training a SVM classifier on that dataset. Plot the decision boundary and test errors.

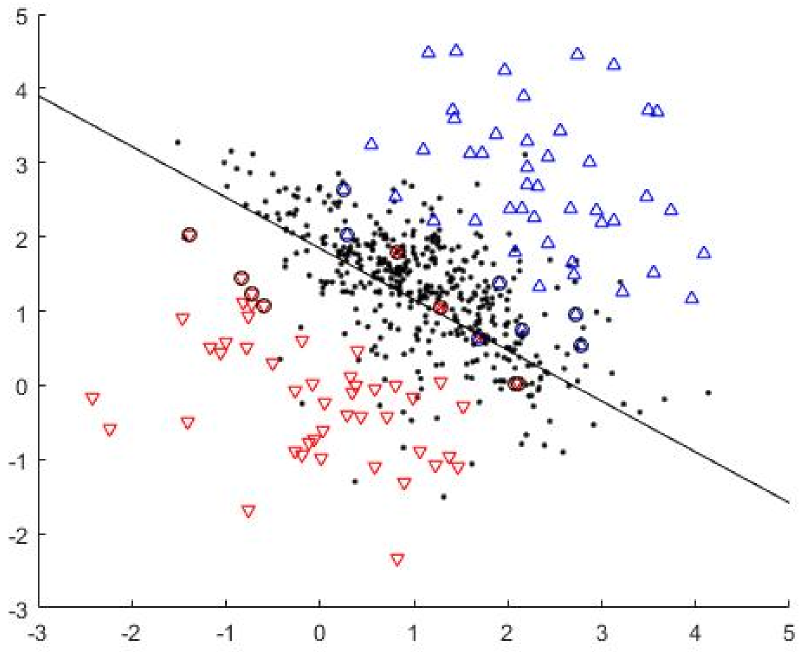
(Run the setup code to clean up and initialize all variables as shown in *Appendix 9: Setup*)

Data Set 1

```
% Rationale:
% The best kernel is the linear one as the data set clearly shows a linear
% decision boundary separating the data. In these cases, it is best to
% avoid overfitting, hence choosing the simplest model. The error rates on
% the test examples also support this claim as the linear decision boundary
% gave the lowest amounts of misclassified testing data (0.0446 of test
% examples misclassified).
```

(Refer to *Appendix 10: Data Set 1* for code to generate the next three plots and testing results)

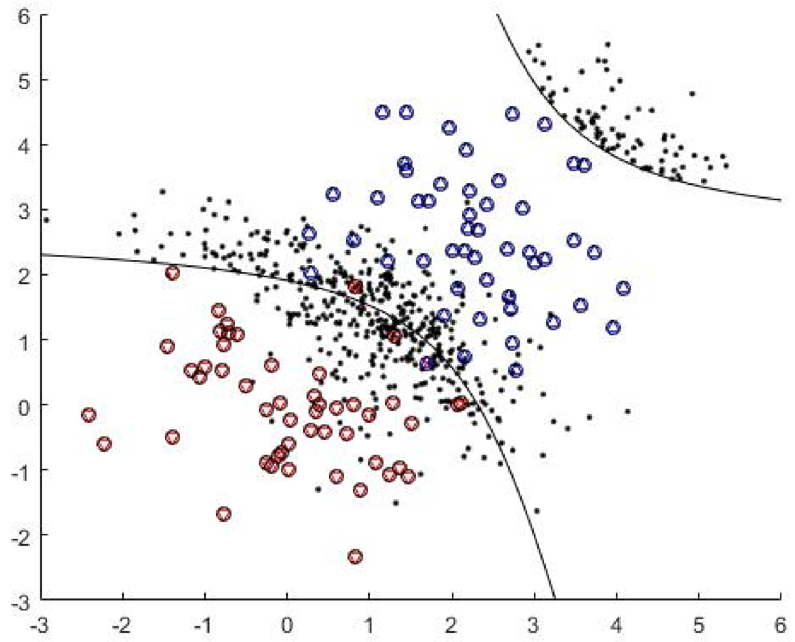
Linear Kernel:



Set 1 – Linear Kernel

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

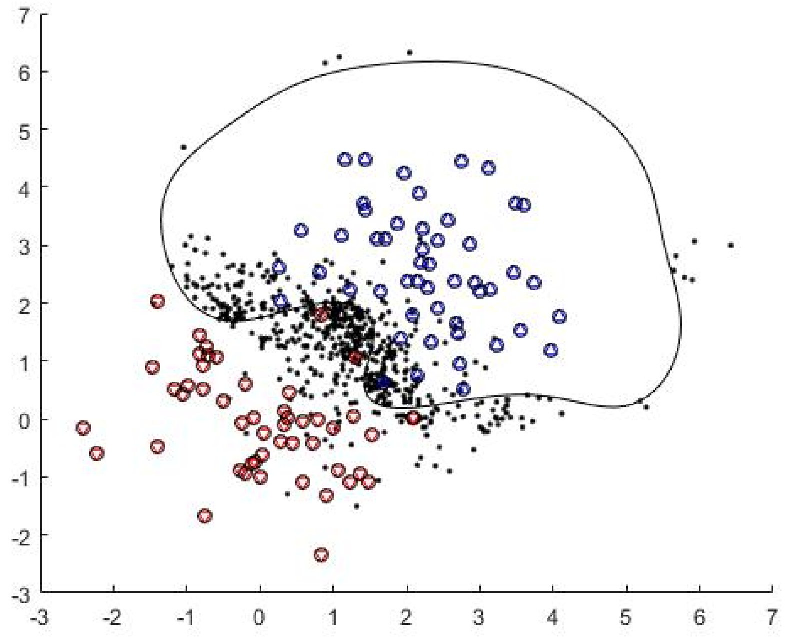
Polynomial Kernel:



Set 1 – Polynomial Kernel

WARNING: 2 training examples were misclassified!!!
TEST RESULTS: 0.0514 of test examples were misclassified.

Gaussian Kernel:



Set 1 – Gaussian Kernel

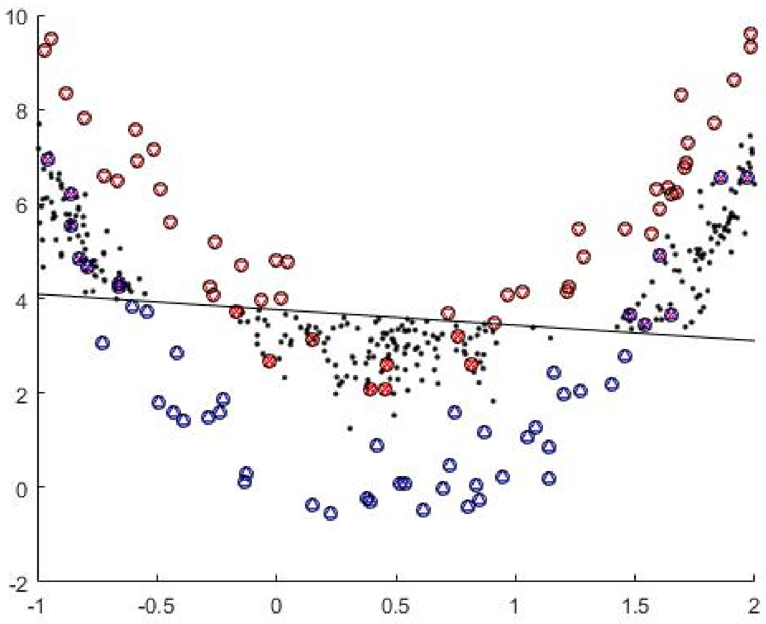
TEST RESULTS: 0.0571 of test examples were misclassified.

Data Set 2

```
% Rationale:
% The best kernel is the second order polynomial one as the data set
% clearly shows a parabola of degree 2 separating the data. In this case,
% the linear kernel would result in being under-fitting, whereas the
% Gaussian kernel would over-fit. The error rates on the test examples also
% support this claim as the polynomial kernel of degree 2 gave the lowest
% amounts of misclassified testing data (0.011 of test examples were
% misclassified).
```

(Refer to *Appendix 11: Data Set 2* for code to generate the next three plots and testing results)

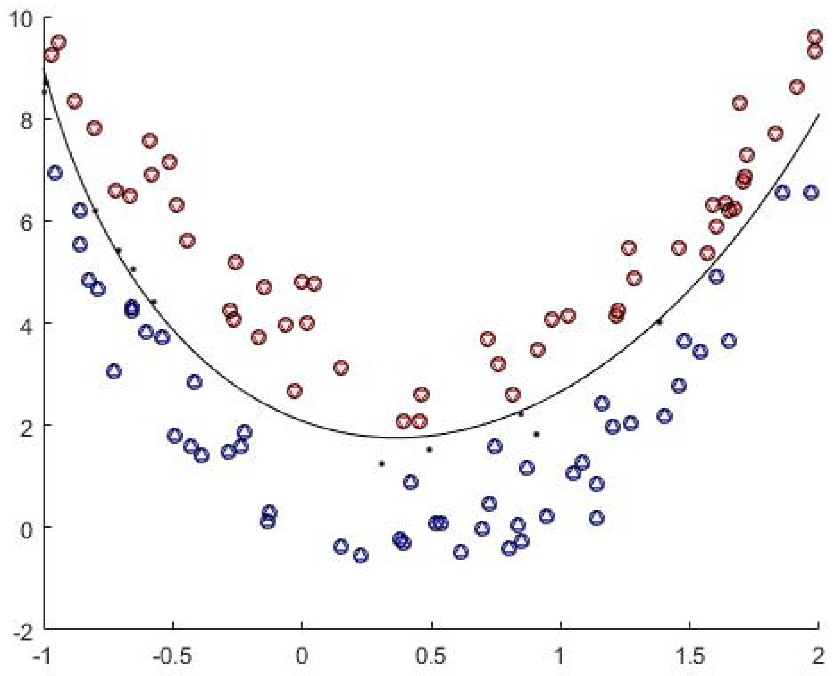
Linear Kernel:



Set 2 – Linear Kernel

WARNING: 21 training examples were misclassified!!!
TEST RESULTS: 0.273 of test examples were misclassified.

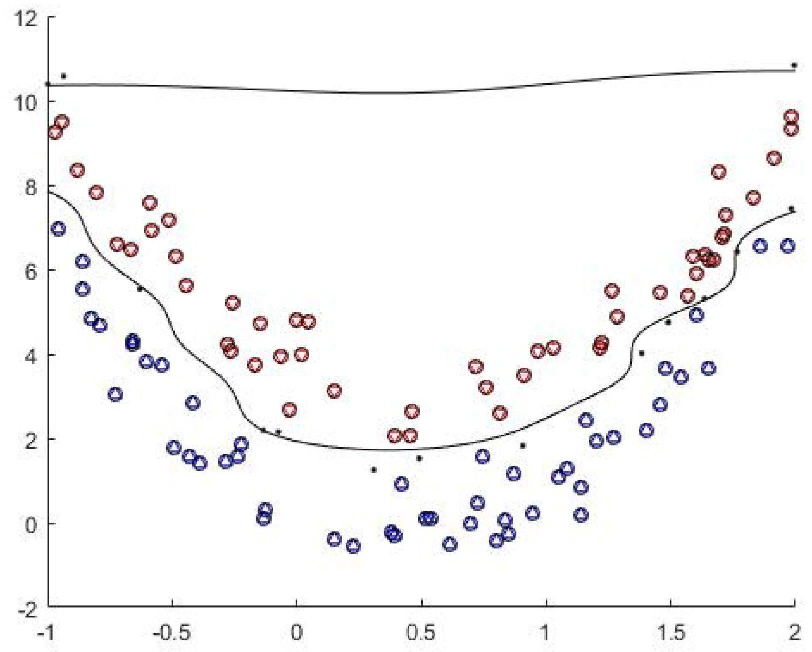
Polynomial Kernel:



Set 2 – Polynomial Kernel

TEST RESULTS: 0.011 of test examples were misclassified.

Gaussian Kernel:



Set 2 – Gaussian Kernel

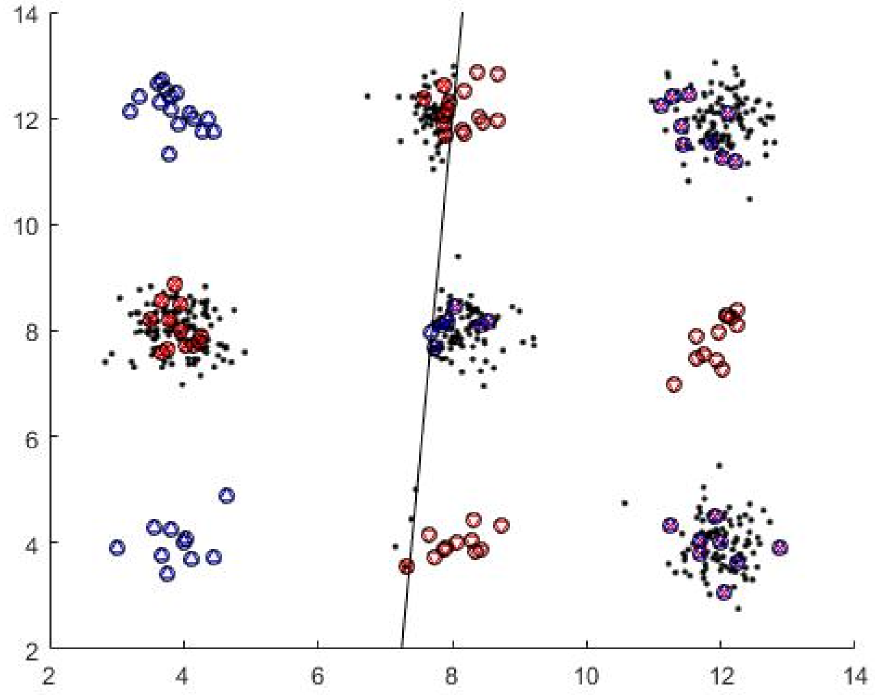
TEST RESULTS: 0.014 of test examples were misclassified.

Data Set 3

```
% Rationale:
% The best kernel is the Gaussian one as the data set clearly shows groups
% of data points clustered, making it not separable with the linear or
% polynomial kernel. In this case, the Gaussian kernel is able to separate
% these groups, hence it is the best choice. The error rates on the test
% examples also support this claim as the Gaussian kernel produced no
% amounts of misclassified testing data.
```

(Refer to *Appendix 12: Data Set 3* for code to generate the next three plots and testing results)

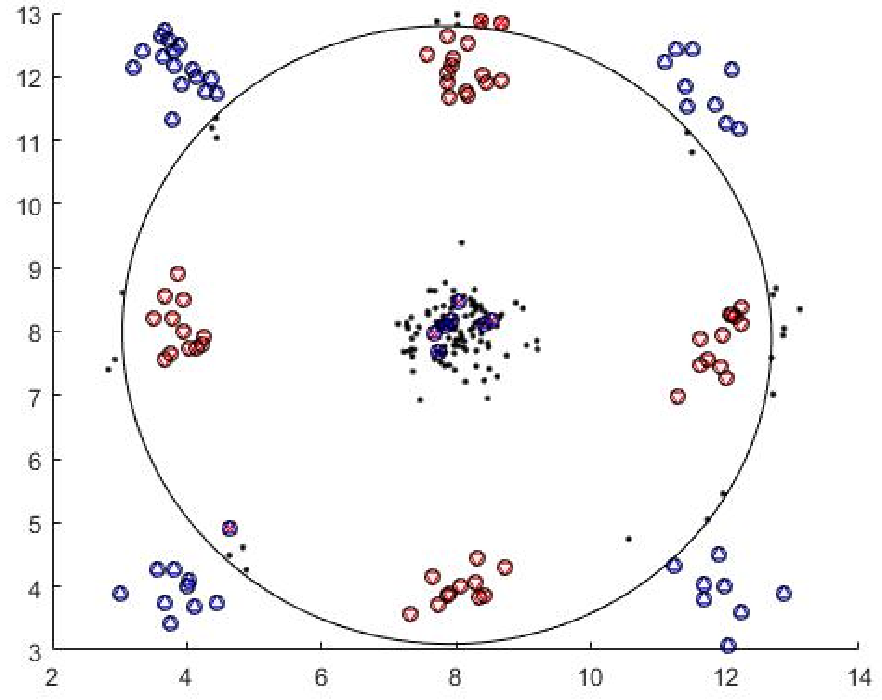
Linear Kernel:



Set 3 – Linear Kernel

WARNING: 43 training examples were misclassified!!!
TEST RESULTS: 0.471 of test examples were misclassified.

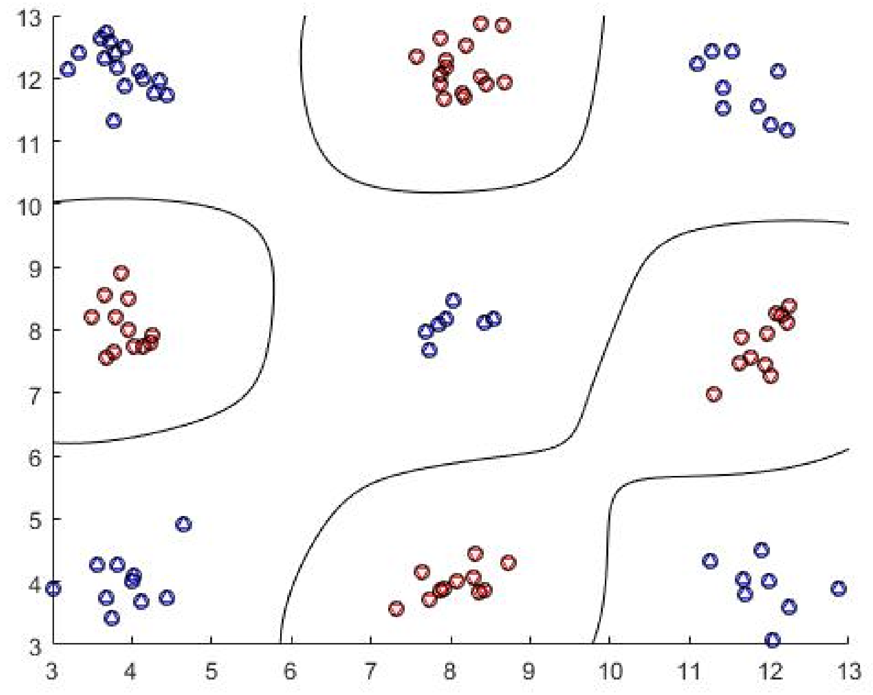
Polynomial Kernel:



Set 3 – Polynomial Kernel

WARNING: 10 training examples were misclassified!!!
TEST RESULTS: 0.132 of test examples were misclassified.

Gaussian Kernel:



Set 3- Gaussian Kernel

TEST RESULTS: 0 of test examples were misclassified.

2. For the digit dataset (set4_train and set4_test), train and test SVM's with a linear, polynomial of degree 2, and Gaussian of standard deviation 1.5 kernels. Report the test errors.

(Refer to *Appendix 13: SVM Problem 2 – Data Set 4* for code to generate the following test errors)
(Refer to *Appendix 14: svm_test_digital.m code* for the code of the function used in *Appendix 13*)

Data Set 4

Linear Kernel:

```
% TEST RESULTS: 0.14 of test examples were misclassified.
```

Polynomial Kernel:

```
% TEST RESULTS: 0.12 of test examples were misclassified.
```

Gaussian Kernel:

```
% TEST RESULTS: 0.085 of test examples were misclassified.
```

```
% Conclusion:
% The linear kernel provided results of 0.14 misclassified test examples.
% The second order polynomial kernel provided results of 0.12 misclassified
% test examples.
% The Gaussian kernel provided results of 0.085 misclassified test
% examples.
```

Bayes Classifiers

a) **To create a joint Bayes classifier, determine how probable it is for the class given its features. The total probability must also sum to one.**

$$P(y = 0 \mid x_1 = 0, x_2 = 0) = \frac{1}{4}$$
$$P(y = 1 \mid x_1 = 0, x_2 = 0) = \frac{3}{4}$$

$$P(y = 0 \mid x_1 = 0, x_2 = 1) = \frac{1}{4}$$
$$P(y = 1 \mid x_1 = 0, x_2 = 1) = \frac{3}{4}$$

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

$$P(y = 0 \mid x_1 = 1, x_2 = 1) = \frac{3}{5}$$
$$P(y = 1 \mid x_1 = 1, x_2 = 1) = \frac{2}{5}$$

Classifying the test set:

x_1	x_2	$P(y = 0 \mid x)$	$P(y = 1 \mid x)$	p̂	Test Result (y)
0	1	$\frac{1}{4}$ or 25%	$\frac{3}{4}$ or 75%	1	1
1	0	$\frac{3}{3}$ or 100%	$\frac{0}{3}$ or 0%	0	1
1	1	$\frac{3}{5}$ or 60%	$\frac{2}{5}$ or 40%	0	0

b) To create a naïve Bayes classifier, first provide the probabilities of each class ($\boldsymbol{P(y)}$)

$$P(y=0)=\frac{8}{16}=\frac{1}{2}$$
$$P(y=1)=\frac{8}{16}=\frac{1}{2}$$

Then provide probabilities of each feature given the class:

$$P(x_1=1\mid y=0)=\frac{6}{8}$$
$$P(x_1=1\mid y=1)=\frac{2}{8}$$
$$P(x_2=1\mid y=0)=\frac{4}{8}$$
$$P(x_2=1\mid y=1)=\frac{5}{8}$$

Then provide probabilities of each feature combination given the class, in the formula:

$$\boldsymbol{P(x|y)=P(x_1|y)*P(x_2|y)}$$

$\boldsymbol{x_1=0,x_2=0}$:

$$\begin{aligned}P(x_1=0,x_2=0|y=0)&=P(x_1=0|y=0)*P(x_2=0|y=0)\\&=\left(1-\frac{6}{8}\right)*\left(1-\frac{4}{8}\right)\\&=\frac{1}{8}\end{aligned}$$

$$\begin{aligned}P(x_1=0,x_2=0|y=1)&=P(x_1=0|y=1)*P(x_2=0|y=1)\\&=\left(1-\frac{2}{8}\right)*\left(1-\frac{5}{8}\right)\\&=\frac{9}{32}\end{aligned}$$

$\boldsymbol{x_1=0,x_2=1}$:

$$\begin{aligned}P(x_1=0,x_2=1|y=0)&=P(x_1=0|y=0)*P(x_2=1|y=0)\\&=\left(1-\frac{6}{8}\right)*\left(\frac{4}{8}\right)\\&=\frac{1}{8}\end{aligned}$$

$$\begin{aligned}P(x_1=0,x_2=1|y=1)&=P(x_1=0|y=1)*P(x_2=1|y=1)\\&=\left(1-\frac{2}{8}\right)*\left(\frac{5}{8}\right)\\&=\frac{15}{32}\end{aligned}$$

$\boldsymbol{x_1=1,x_2=0}$:

$$\begin{aligned}P(x_1=1,x_2=0|y=0)&=P(x_1=1|y=0)*P(x_2=0|y=0)\\&=\left(\frac{6}{8}\right)*\left(1-\frac{4}{8}\right)\\&=\frac{3}{8}\end{aligned}$$

$$\begin{aligned}P(x_1=1,x_2=0|y=1)&=P(x_1=1|y=1)*P(x_2=0|y=1)\\&=\left(\frac{2}{8}\right)*\left(1-\frac{5}{8}\right)\\&=\frac{3}{32}\end{aligned}$$

$\boldsymbol{x_1 = 1, x_2 = 1}$:

$$\begin{aligned} P(x_1 = 1, x_2 = 1|y = 0) &= P(x_1 = 1|y = 0) * P(x_2 = 1|y = 0) \\ &= \left(\frac{6}{8}\right) * \left(\frac{4}{8}\right) \\ &= \frac{3}{8} \end{aligned}$$

$$\begin{aligned} P(x_1 = 1, x_2 = 1|y = 1) &= P(x_1 = 1|y = 1) * P(x_2 = 1|y = 1) \\ &= \left(\frac{2}{8}\right) * \left(\frac{5}{8}\right) \\ &= \frac{5}{32} \end{aligned}$$

Then provide probabilities of $\boldsymbol{x}$ ($\boldsymbol{P(x)}$) where $\boldsymbol{P(x) = \sum_{y'}(P(x|y') * P(y'))}$

$$\begin{aligned} P(x_1 = 0, x_2 = 0) &= \left(\frac{1}{8} * \frac{1}{2}\right) + \left(\frac{9}{32} * \frac{1}{2}\right) \\ &= \frac{1}{16} + \frac{9}{64} \\ &= \frac{13}{64} \end{aligned}$$

$$\begin{aligned} P(x_1 = 0, x_2 = 1) &= \left(\frac{1}{8} * \frac{1}{2}\right) + \left(\frac{15}{32} * \frac{1}{2}\right) \\ &= \frac{1}{16} + \frac{15}{64} \\ &= \frac{19}{64} \end{aligned}$$

$$\begin{aligned} P(x_1 = 1, x_2 = 0) &= \left(\frac{3}{8} * \frac{1}{2}\right) + \left(\frac{3}{32} * \frac{1}{2}\right) \\ &= \frac{3}{16} + \frac{3}{64} \\ &= \frac{15}{64} \end{aligned}$$

$$\begin{aligned} P(x_1 = 1, x_2 = 1) &= \left(\frac{3}{8} * \frac{1}{2}\right) + \left(\frac{5}{32} * \frac{1}{2}\right) \\ &= \frac{3}{16} + \frac{5}{64} \\ &= \frac{17}{64} \end{aligned}$$

Finally, calculate the probability of a class through the Bayesian classification

$$\boldsymbol{P(y|x) = \frac{P(x|y) * P(y)}{P(x)}}$$

$\boldsymbol{x_1 = 0, x_2 = 0}$:

$$\begin{aligned} P(y = 0|x_1 = 0, x_2 = 0) &= \frac{P(x_1 = 0, x_2 = 0|y = 0) * P(y = 0)}{P(x_1 = 0, x_2 = 0)} \\ &= \frac{\frac{1}{8} * \frac{1}{2}}{\frac{13}{64}} \\ &\approx 31\% \end{aligned}$$

$$\therefore P(y = 1|x_1 = 0, x_2 = 0) \approx 69\%$$

$\boldsymbol{x_1 = 0, x_2 = 1}$:

$$\begin{aligned} P(y = 0|x_1 = 0, x_2 = 1) &= \frac{P(x_1 = 0, x_2 = 1|y = 0) * P(y = 0)}{P(x_1 = 0, x_2 = 1)} \\ &= \frac{\frac{1}{8} * \frac{1}{2}}{\frac{19}{64}} \\ &\approx 21\% \end{aligned}$$

$$\therefore P(y = 1|x_1 = 0, x_2 = 1) \approx 79\%$$

$\boldsymbol{x_1 = 1, x_2 = 0}$:

$$P(y = 0|x_1 = 1, x_2 = 0) = \frac{P(x_1 = 1, x_2 = 0|y = 0) * P(y = 0)}{P(x_1 = 1, x_2 = 0)}$$
$$= \frac{\frac{3}{8} * \frac{1}{2}}{\frac{15}{64}}$$
$$\approx 80\%$$
$$\therefore P(y = 1|x_1 = 1, x_2 = 0) \approx 20\%$$

$\boldsymbol{x_1 = 1, x_2 = 1}$:

$$P(y = 0|x_1 = 1, x_2 = 1) = \frac{P(x_1 = 1, x_2 = 1|y = 0) * P(y = 0)}{P(x_1 = 1, x_2 = 1)}$$
$$= \frac{\frac{3}{8} * \frac{1}{2}}{\frac{17}{64}}$$
$$\approx 71\%$$
$$\therefore P(y = 1|x_1 = 1, x_2 = 1) \approx 29\%$$

Classifying the test set:

$\boldsymbol{x_1}$	$\boldsymbol{x_2}$	$\boldsymbol{P(y = 0 \mid x)}$	$\boldsymbol{P(y = 1 \mid x)}$	p̂	Test Result (y)
0	1	21%	79%	1	1
1	0	80%	20%	0	1
1	1	71%	29%	0	0

Part B: PCA & Clustering

Eigen Faces

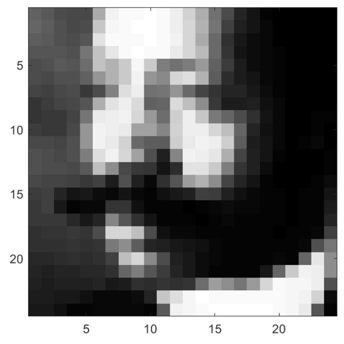


Figure 1. Example face (face 1)

(a) Subtract the mean of the face to make data zero-mean. Take the SVD of data.

```
mu = mean(X,1); % mean of each feature
X0 = bsxfun(@minus, X, mu); % Making X zero-mean

[U, S, V] = svd(X0); % Taking SVD of X

W = U*S; % Used later to compute approximation to X0
```

(b) Computing mean squared error (MSE) of approximations of X_0 for $K = 1 \dots 10$

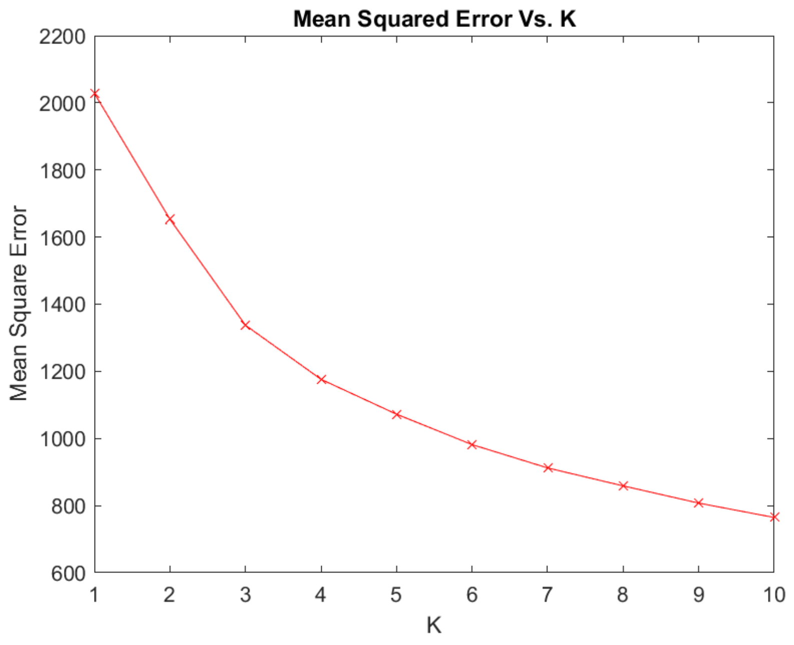


Figure 2. MSE vs. K

Refer to appendix 1 for code used to generate figure from (b)

(c) Displaying first 4 principal directions of the data

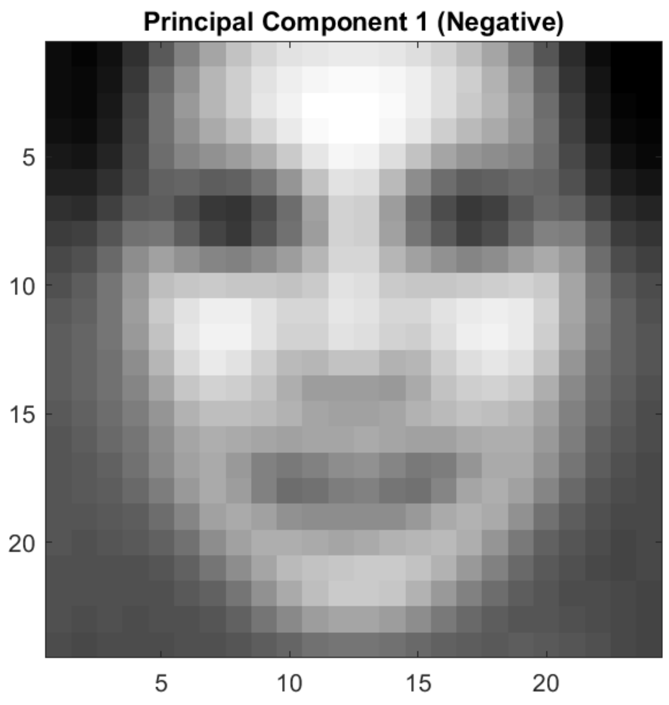


Figure 3. Negative Principal Component 1

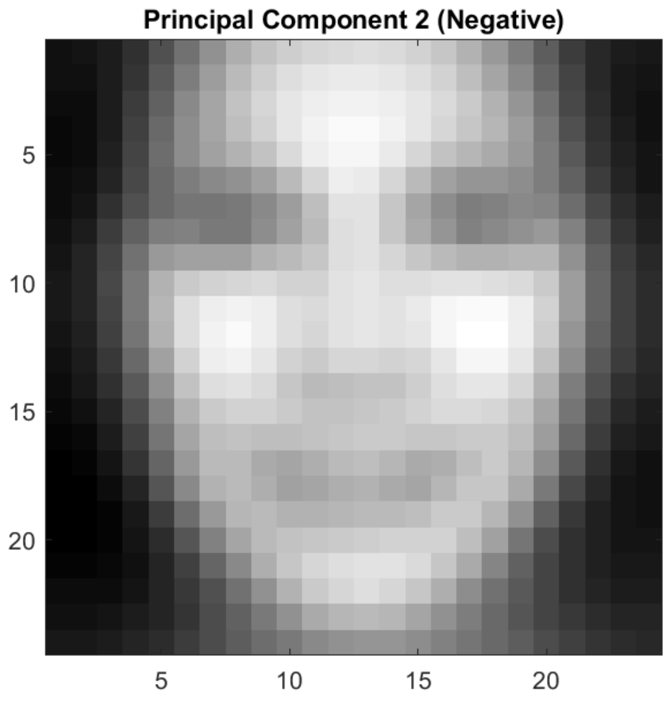


Figure 4. Negative Principal Component 2

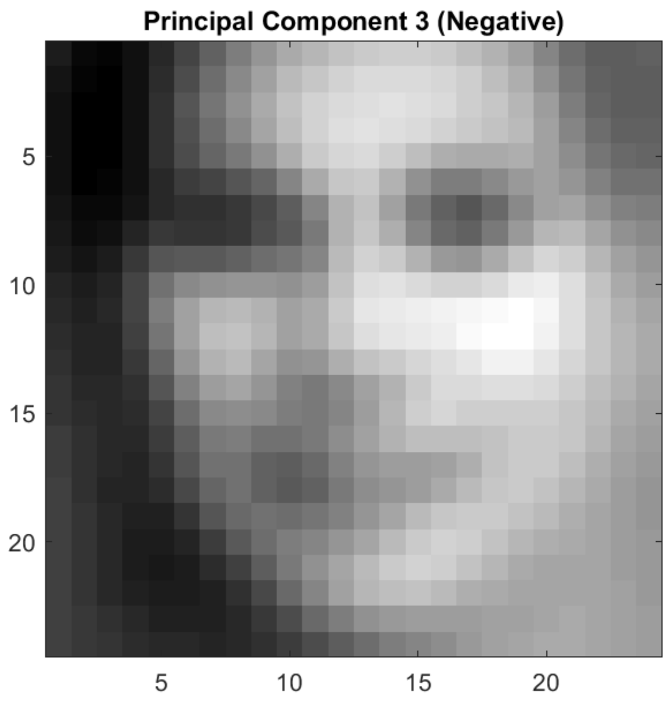


Figure 5. Negative Principal Component 3

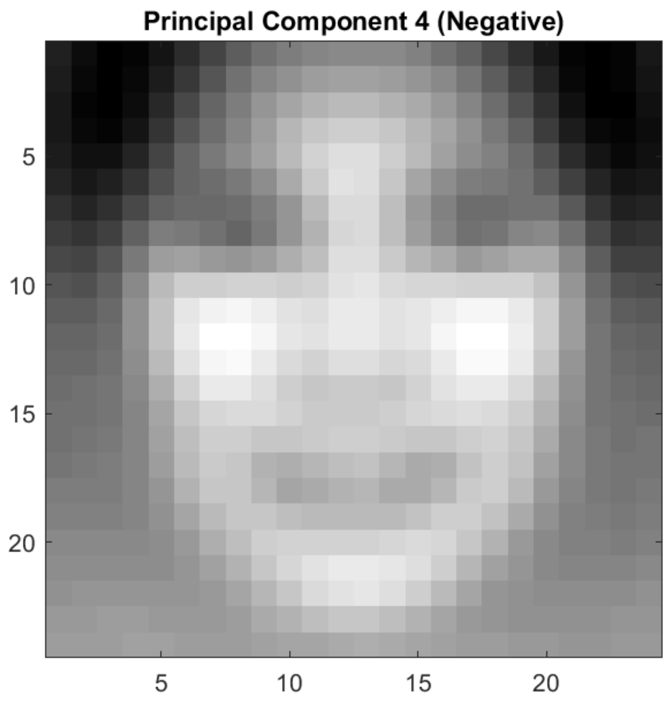


Figure 6. Negative Principal Component 4

Refer to appendix 2.1 for code used to generate the figures.
Refer to appendix 2.2 for additional figures that correspond to $\mu + \alpha\, V(:,j)'$

(d) Latent space visualisation

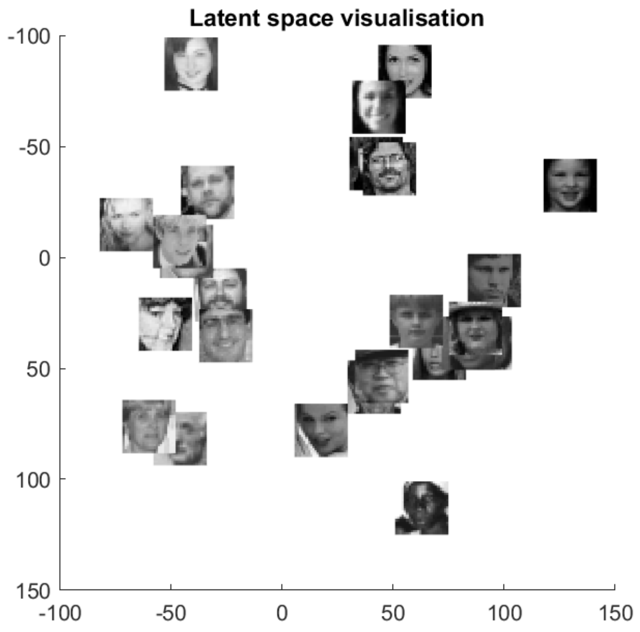


Figure 7. Latent space visualisation

Refer to appendix 3 for code used to generate the figure.

(ei) Reconstruction of face 1000 using only 5, 10 and 50 principal components

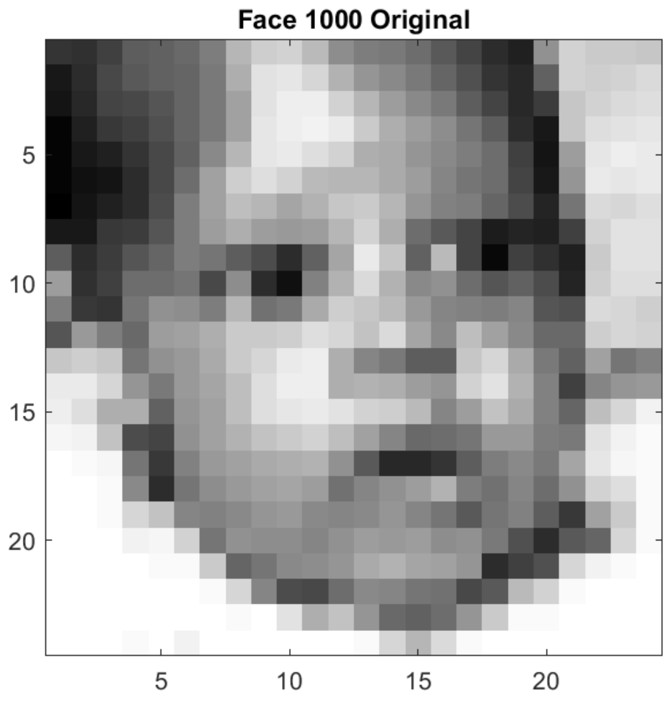


Figure 8. Original Face 1000

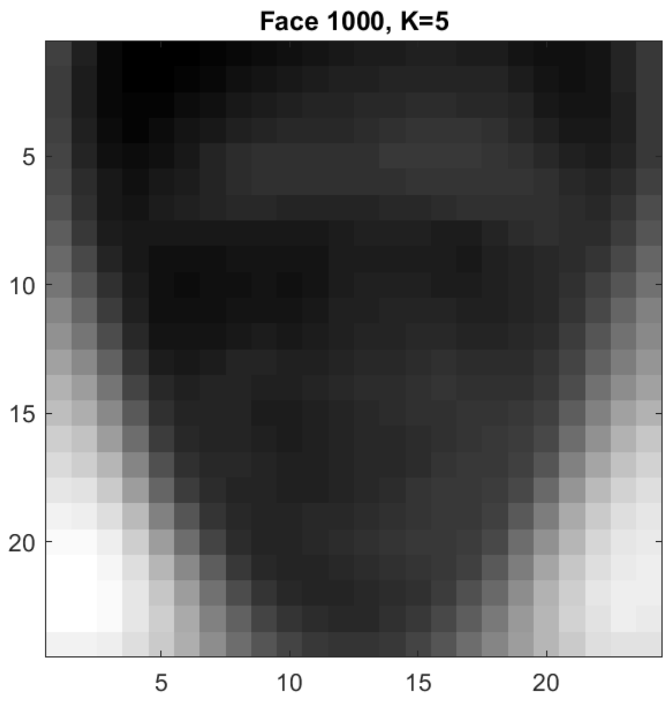


Figure 9. Face 1000 Reconstruction with 5 principal components

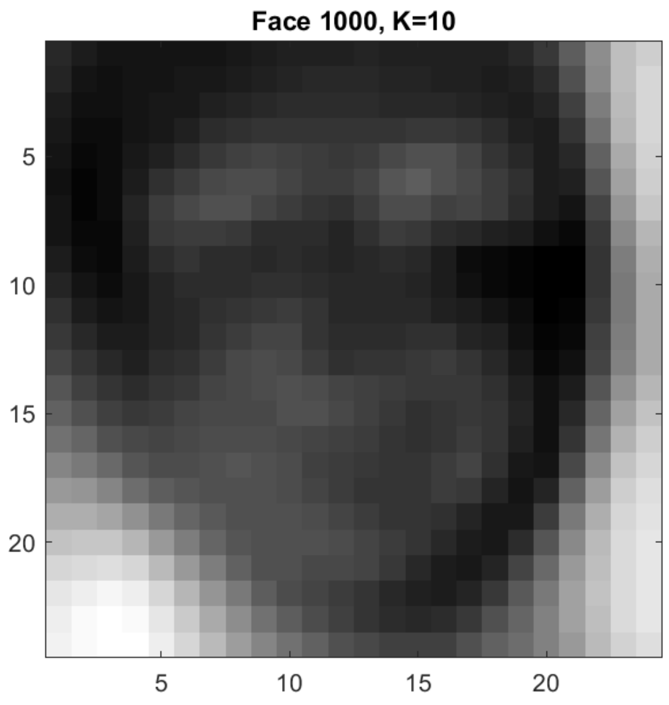


Figure 10. Face 1000 Reconstruction with 10 principal components

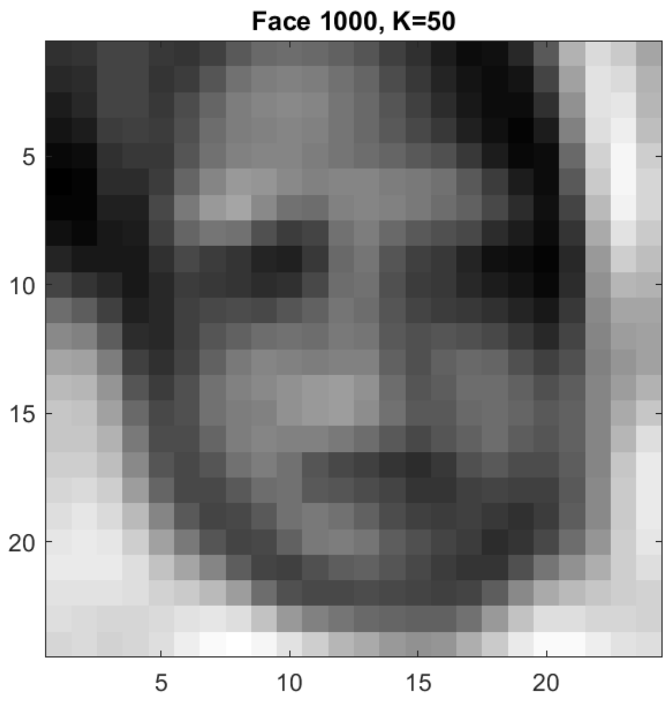


Figure 11. Face 1000 Reconstruction with 50 principal components

(eii) Reconstruction of face 2000 using only 5, 10 and 50 principal components

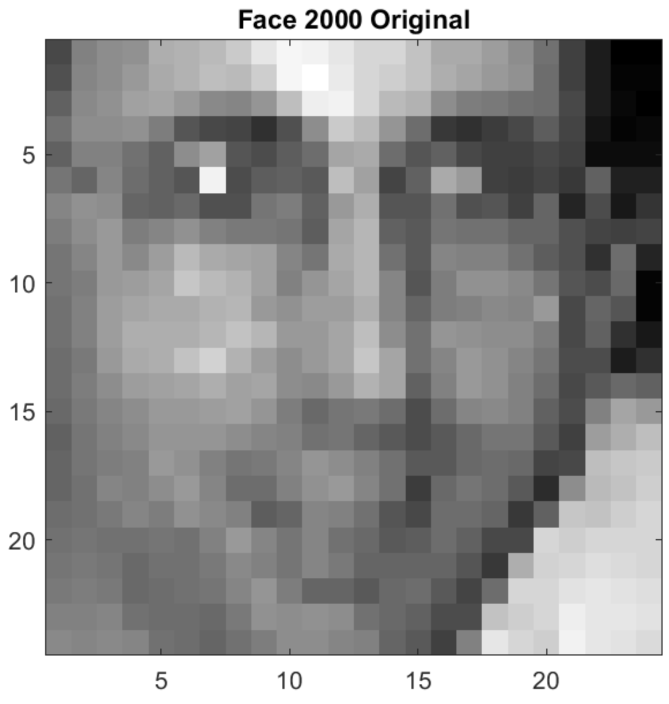


Figure 12. Original Face 2000

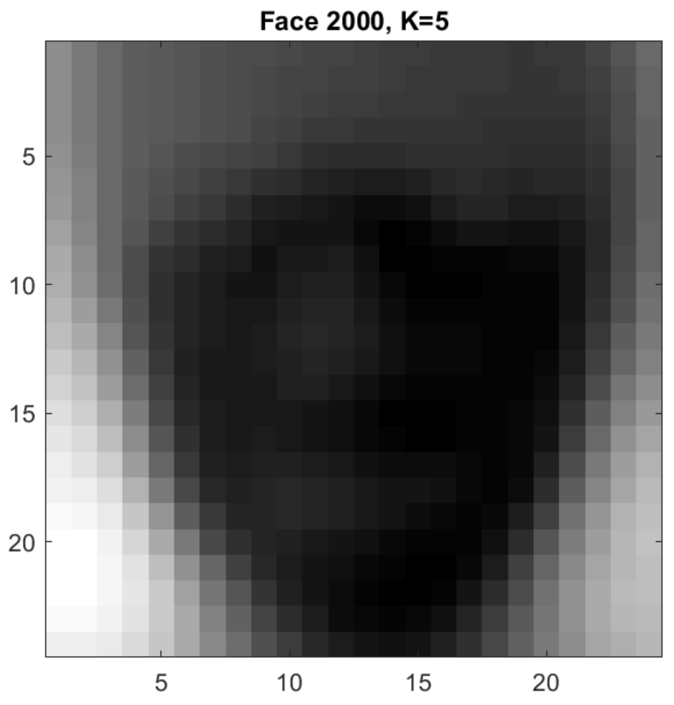


Figure 13. Face 2000 Reconstruction with 5 principal components

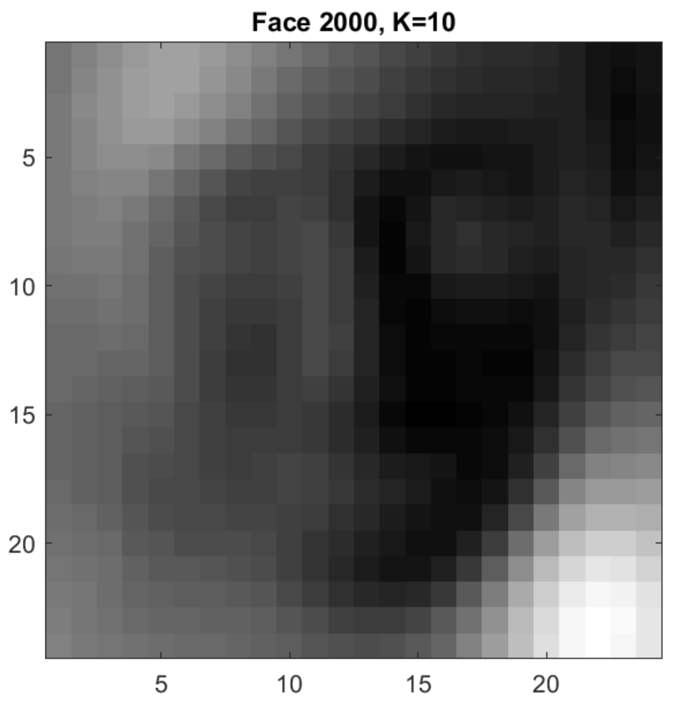


Figure 14. Face 2000 Reconstruction with 10 principal components

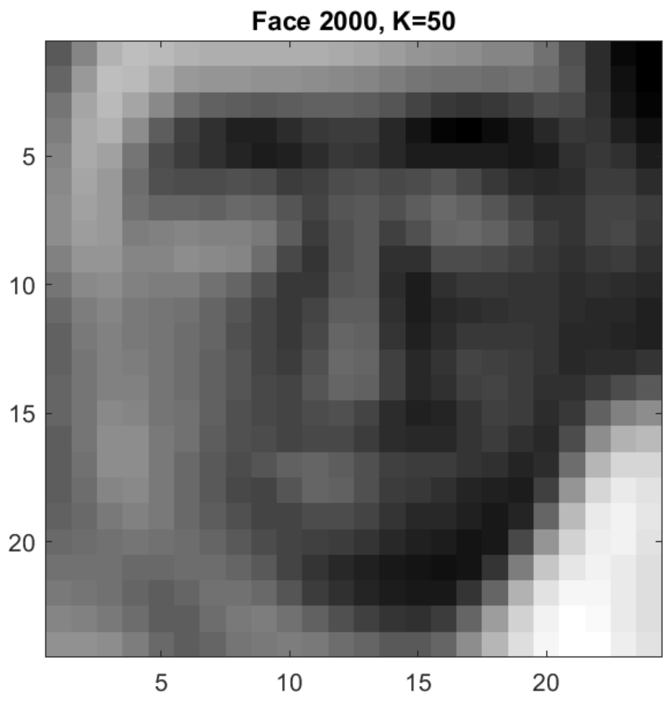


Figure 15. Face 2000 Reconstruction with 50 principal components

Refer to appendix 4 for code used to generate face figures from (ei) and (eii)

Clustering

(a) Load in Iris data and plot it

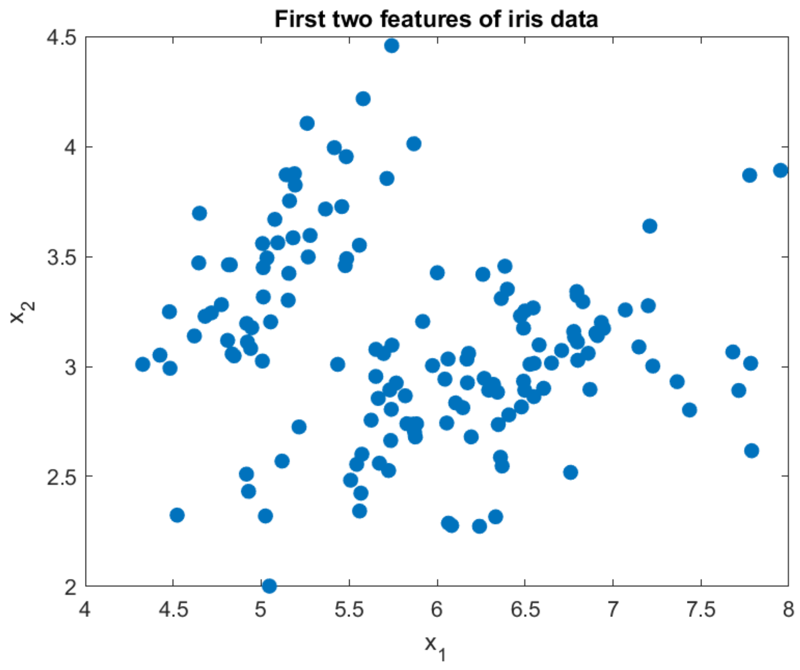


Figure 16. First two features of iris data

Refer to appendix 5 for code used to generate figures from (a)

(b) K-Means on data for k = 5 and k = 20

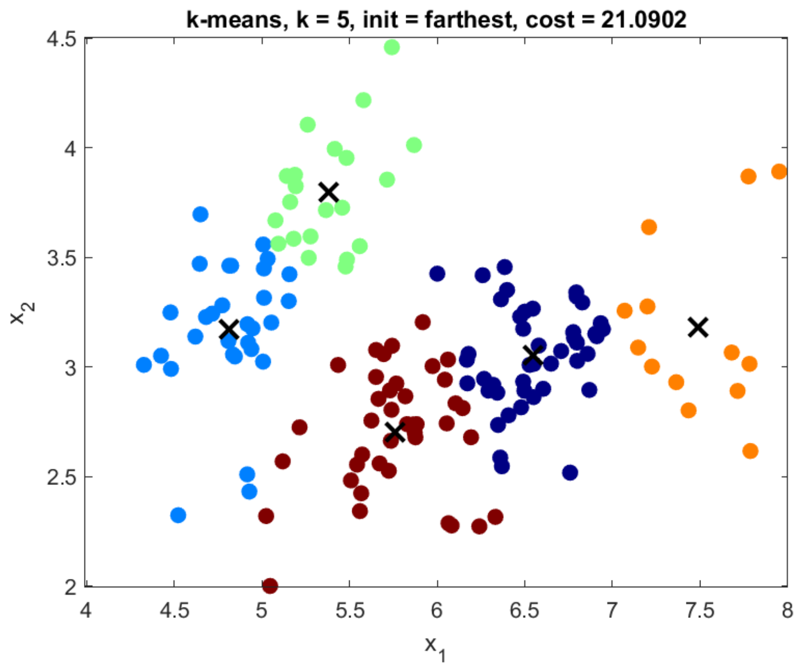


Figure 17. K-Means on iris data with k = 5, initialization = farthest

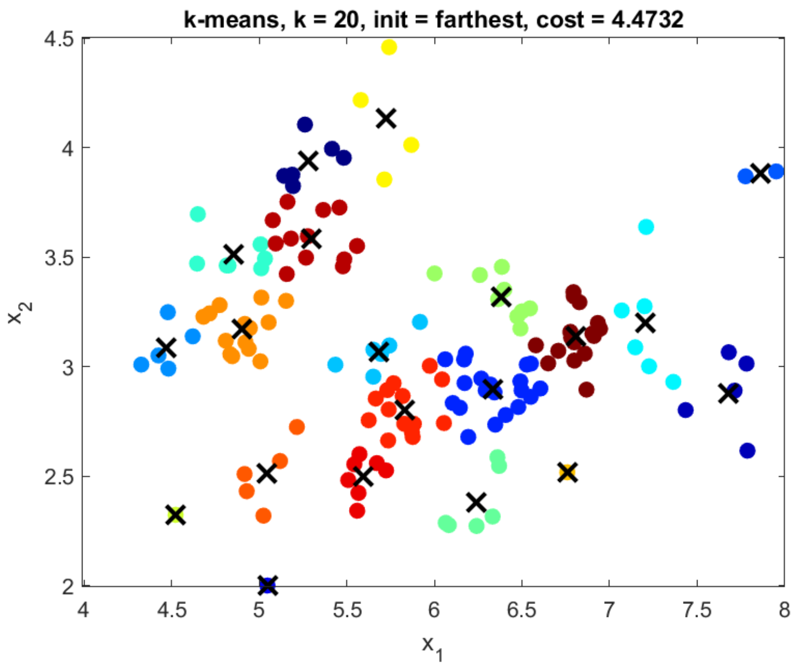


Figure 18. K-Means on iris data with k = 20, initialization = farthest

K-means was run on the iris dataset multiple times with multiple; each time with different initializations. We found that that each initialization resulted in very similar costs but the 'farthest' initialization yielded the lowest cost. In addition, the solutions each initialization found were very similar.

Refer to appendix 6.1 for other initializations.
Refer to appendix 6.2 for code used to generate the figures of (b).

(c) Agglomerative clustering on the data

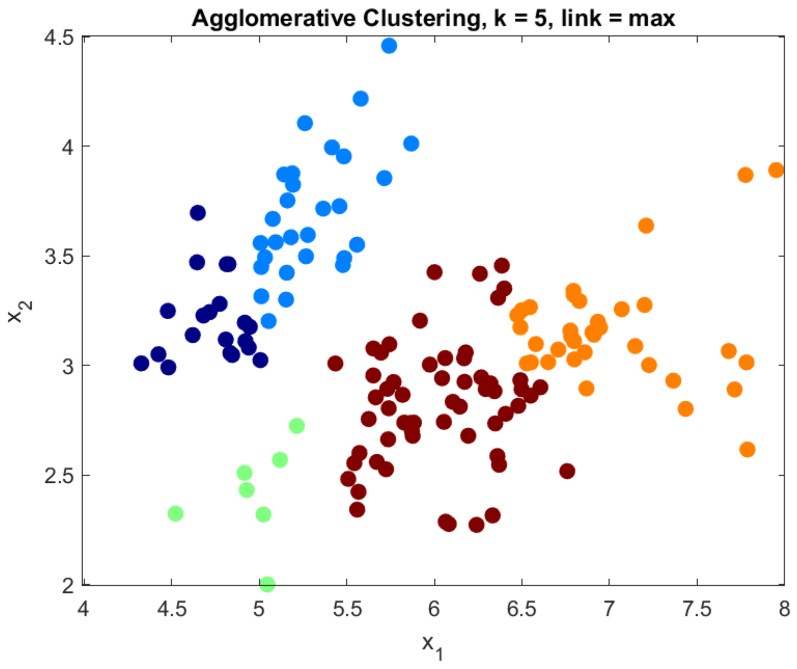


Figure 19. Agglomerative Clustering with 5 clusters and complete linkage

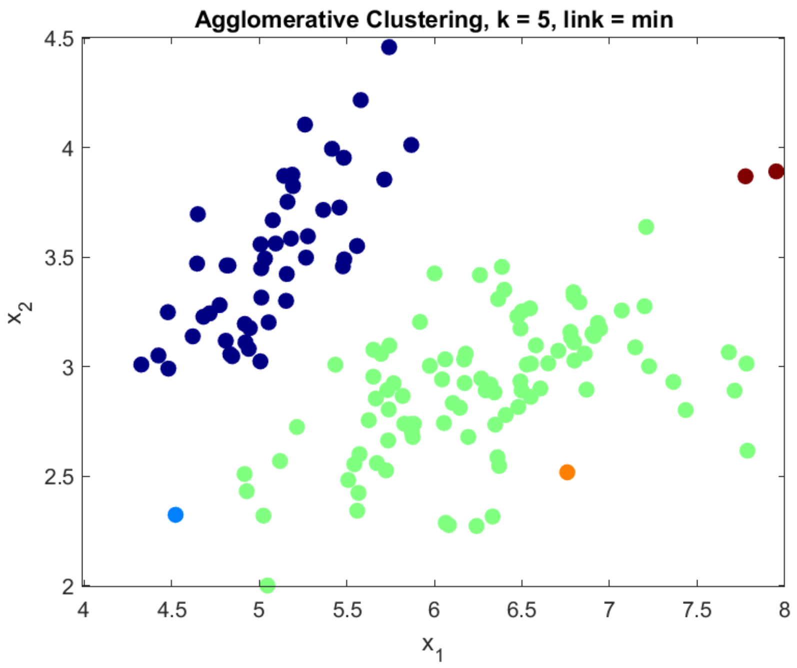


Figure 20. Agglomerative Clustering with 5 clusters and single linkage

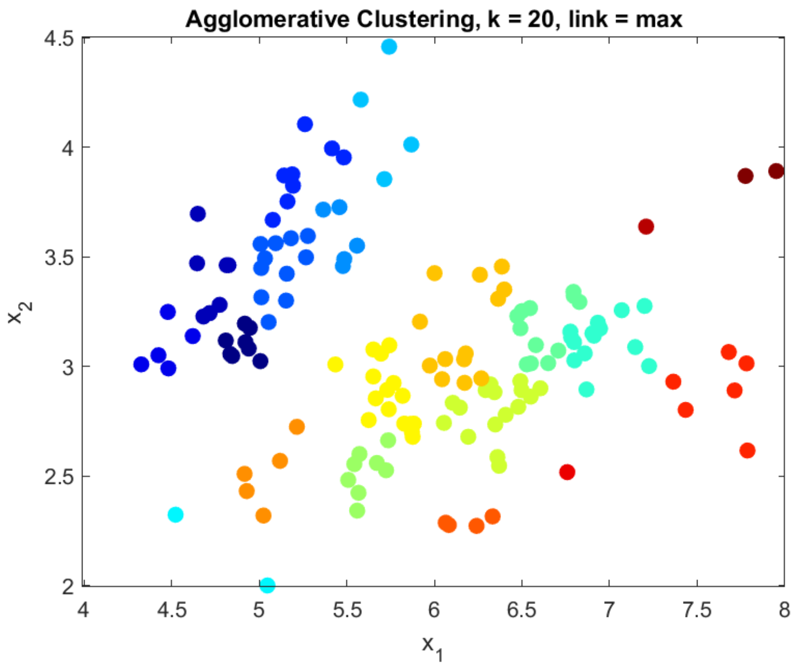


Figure 21. Agglomerative Clustering with 20 clusters and complete linkage

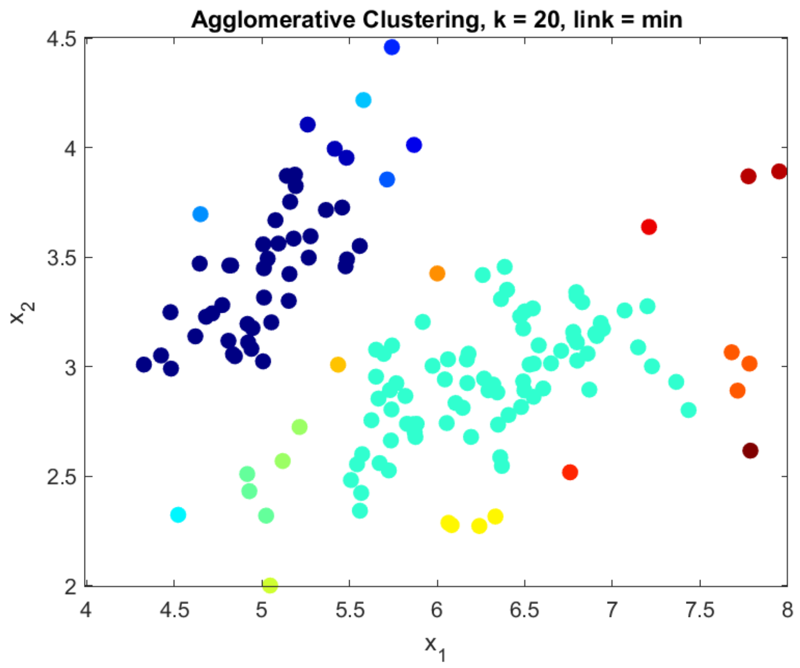


Figure 22. Agglomerative Clustering with 20 clusters and single linkage

Agglomerative clustering with complete linkage yields drastically different results to single linkage. Complete linkage yields similar clustering to k-means while single linkage tends to result in bigger clusters.

Refer to Appendix 7 for code used to generate figures in (c)

(d) EM Gaussian Mixture Model (GMM)

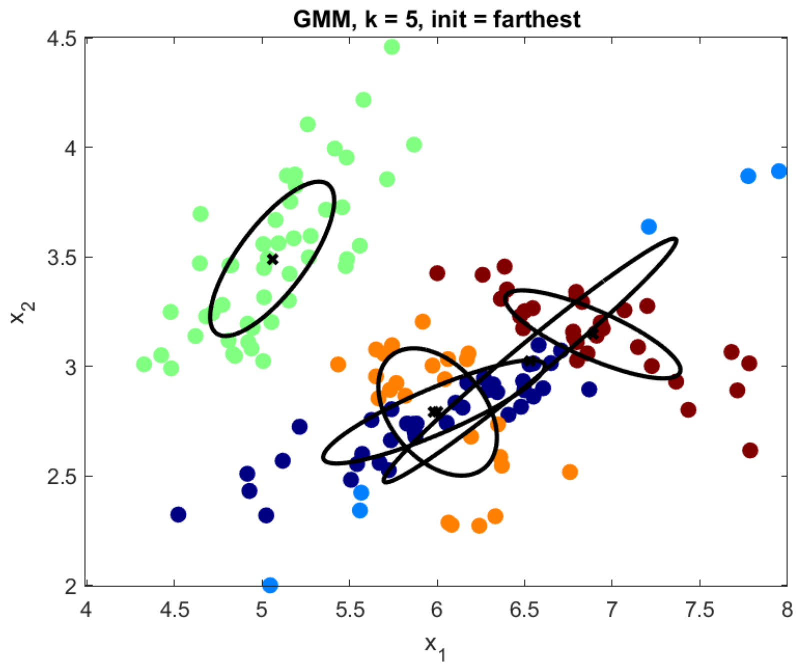


Figure 23. GMM with 5 clusters and 'farthest' initialization

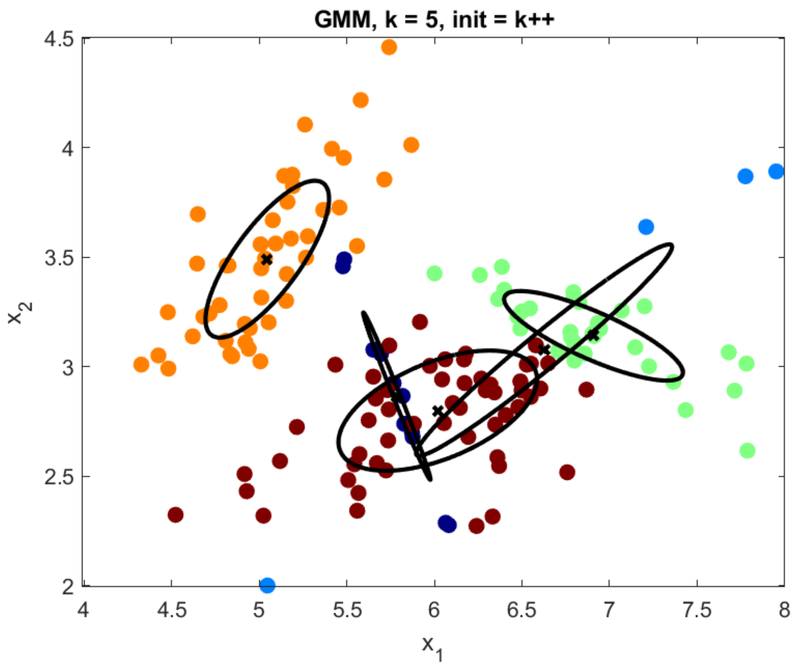


Figure 24. GMM with 5 clusters and 'k++' initialization

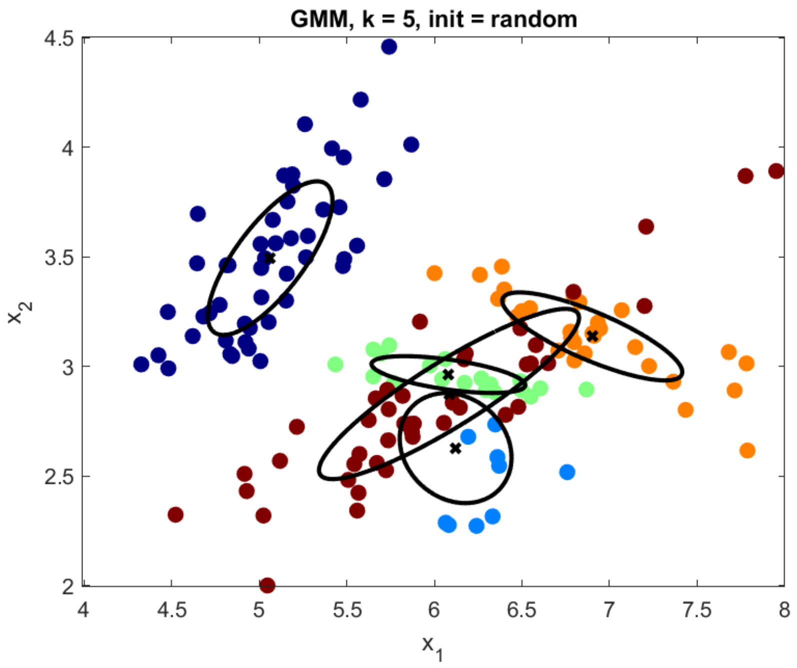


Figure 25. GMM with 5 clusters and 'random' initialization

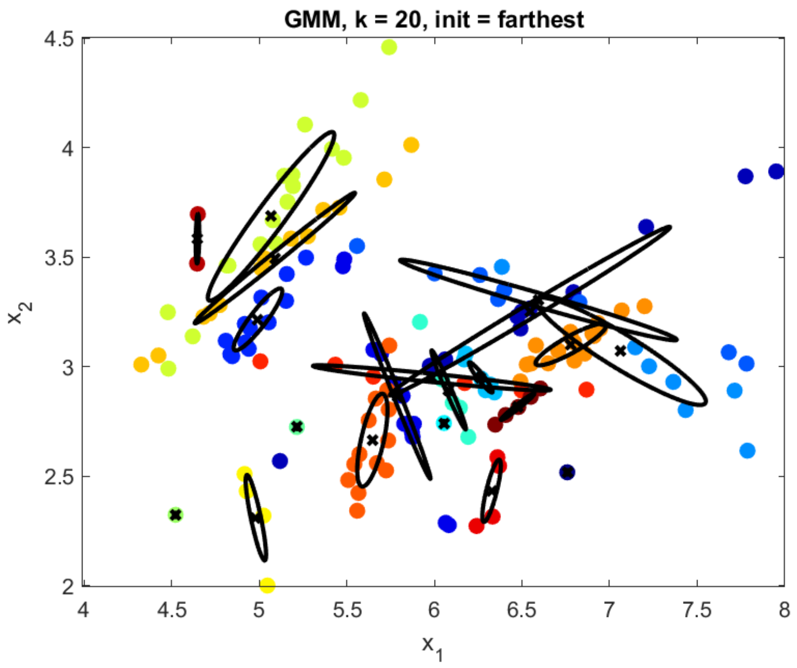


Figure 26. GMM with 20 clusters and 'farthest' initialization

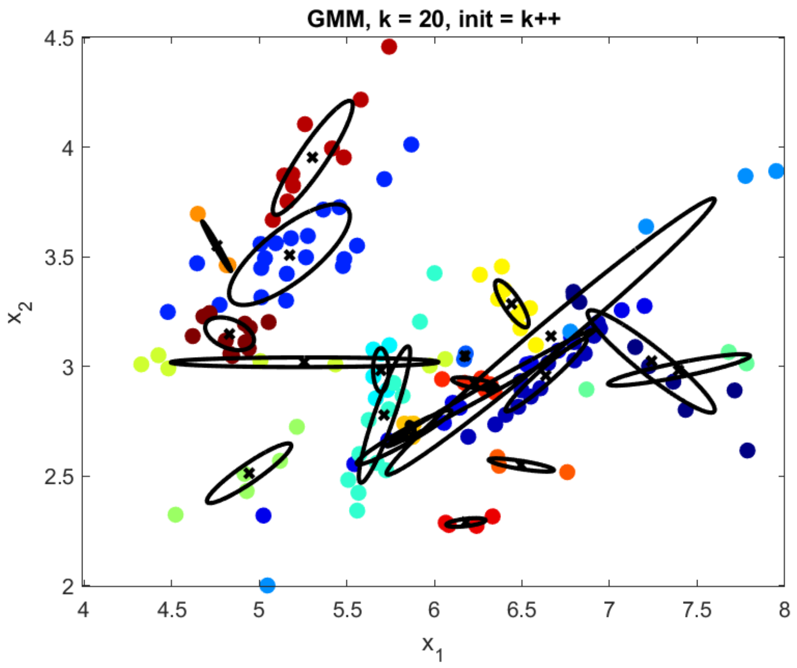


Figure 27. GMM with 20 clusters and 'k++' initialization

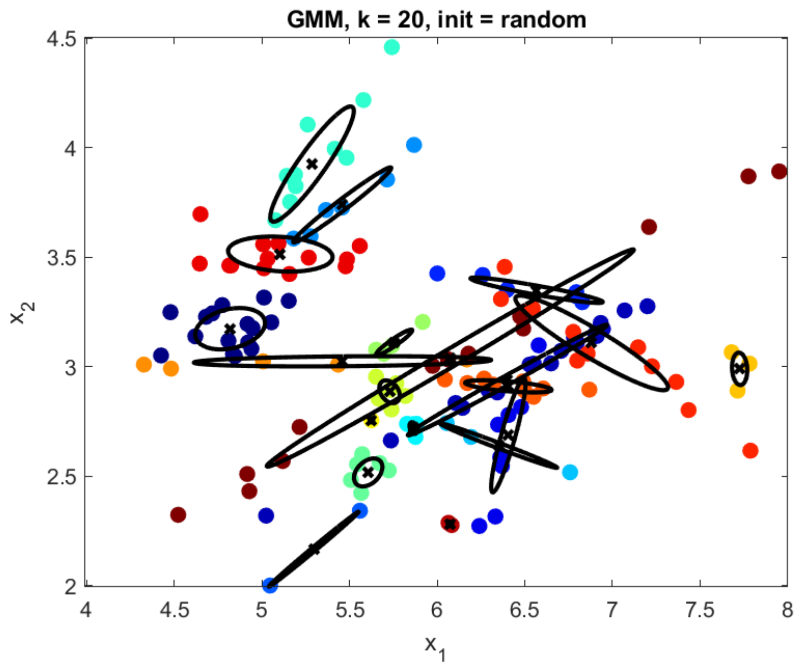


Figure 28. GMM with 20 clusters and 'random' initialization

As the data is not linearly separable between the 3 classes, GMM is much better equipped to cluster this data. GMM clusters the data based on probability which is advantageous for this dataset.

Refer to appendix 8.1 for log likelihood maximization figures.
Refer to appendix 8.2 for code used to generate figures from (d).

Appendices

Code presented in the following appendices may not be formatted correctly. Please refer to the source files for correct formatting.

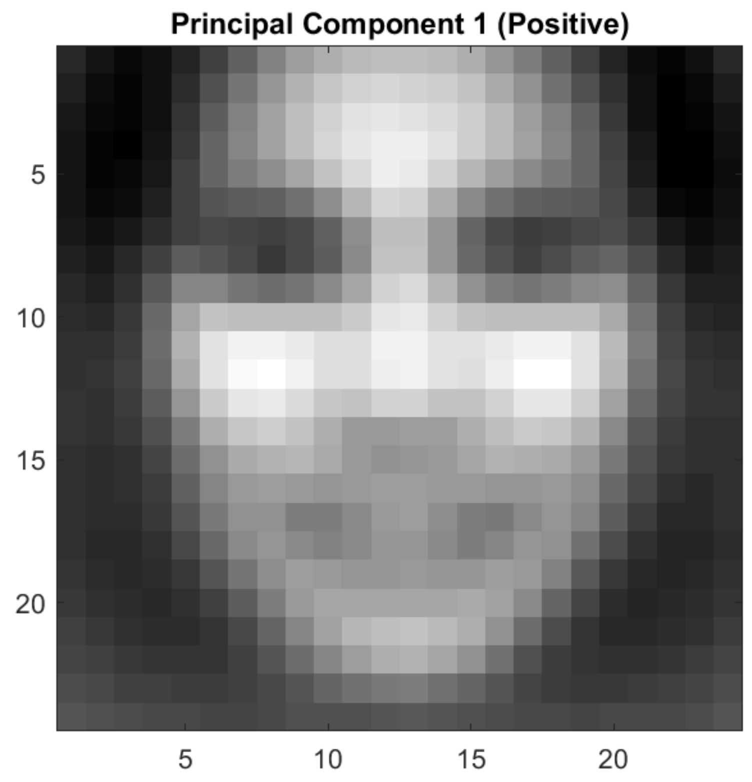
Appendix 1: Code used to generate MSE vs. K figure.

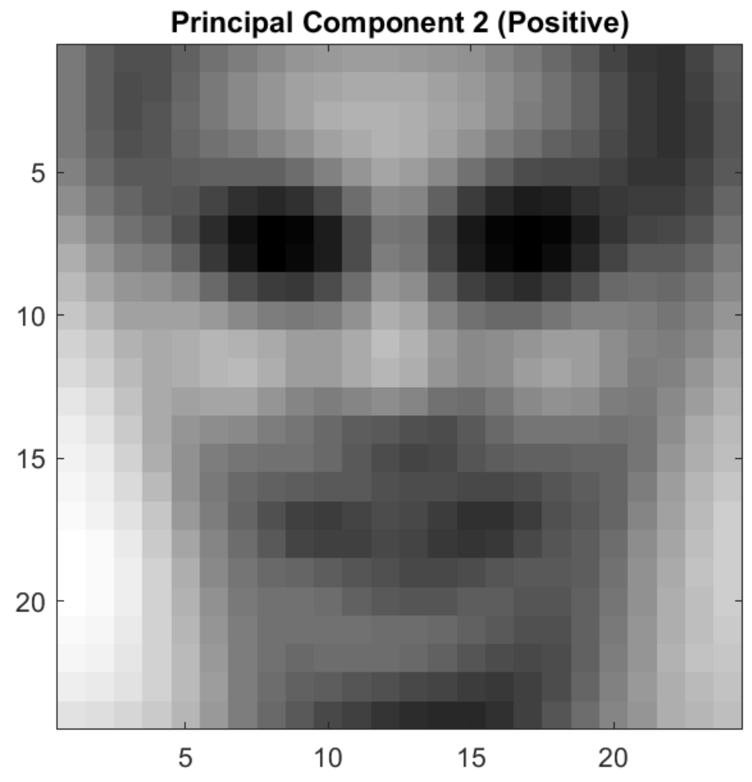
```
K = 1:10;
MSErr = zeros(size(K));
for i = K
    X0h = W(:, 1:i)*V(:, 1:i)';
    MSErr(i) = mean(mean((X0-X0h).^2));
end

FigHandle = plot(K, MSErr, '-rx', 'DisplayName', 'MSErr');

xlabel('K');
ylabel('Mean Square Error');
titleStr = 'Mean Squared Error Vs. K';
title('Mean Squared Error Vs. K');
saveas(FigHandle, [titleStr '.png']);

close all;
```

Appendix 2.1: Code used to generate principal directions of the data.

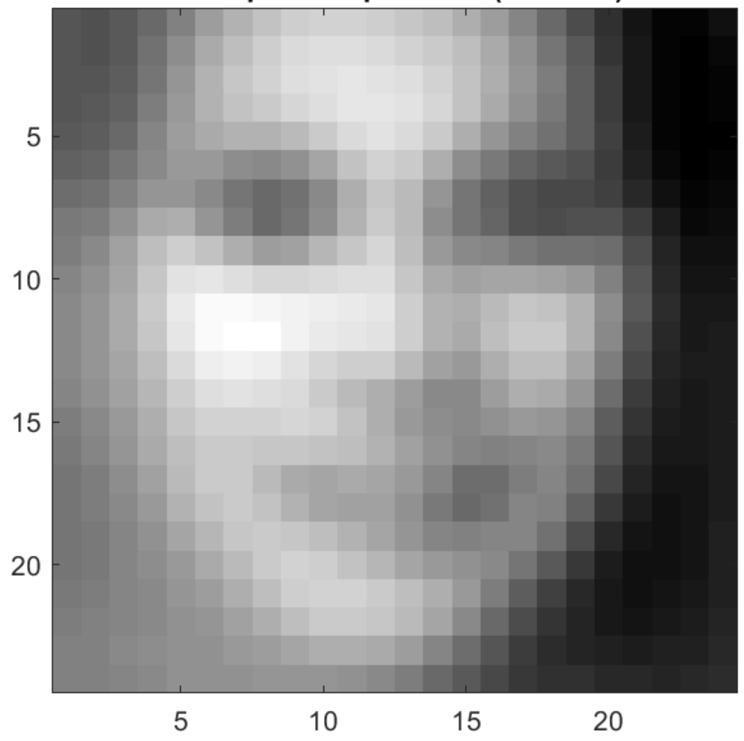
```
for j = 1:4
    alpha = 2*median(abs(W(:,j)));
    posPC = mu + alpha * V(:,j)';
    negPC = mu - alpha * V(:,j)';

    posIMG = reshape(posPC, [24, 24]);
    negIMG = reshape(negPC, [24, 24]);

    FigHandle = figure;
    imagesc(posIMG);
    colormap gray;
    axis square;
    titleStr = ['Principal Component ', num2str(j), ' (Positive)'];
    title(titleStr);
    saveas(FigHandle, [titleStr '.png']);
    close all;

    FigHandle = figure;
    imagesc(negIMG);
    colormap gray;
    axis square;
    titleStr = ['Principal Component ', num2str(j), ' (Negative)'];
    title(titleStr);
    saveas(FigHandle, [titleStr '.png']);
    close all;
end
```

Appendix 2.2: Figures corresponding to $\mu + \alpha\, V(:, j)'$

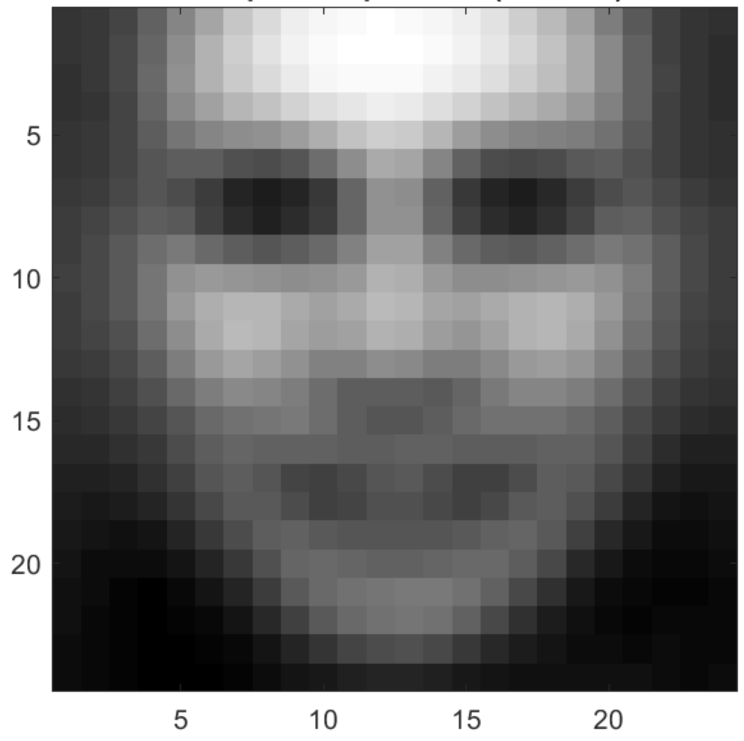




Principal Component 3 (Positive)



Principal Component 4 (Positive)



Appendix 3: Latent space visualisation code

```
randidx = randperm(m);  % Randomize indices
idx = randidx(1:25);    % Get first 25 random indices of faces

FigHandle = figure; hold on; axis ij; colormap(gray);
range = max(W(idx,1:2)) - min(W(idx,1:2)); % find range of coordinates to be plotted
scale = [200 200]./range; % want 24x24 to be visible
for i=idx, imagesc(W(i,1)*scale(1),W(i,2)*scale(2), reshape(X(i,:),24,24)); end
axis square;
titleStr = 'Latent space visualisation';
title(titleStr);
saveas(FigHandle, [titleStr '.png']);
close all;
```

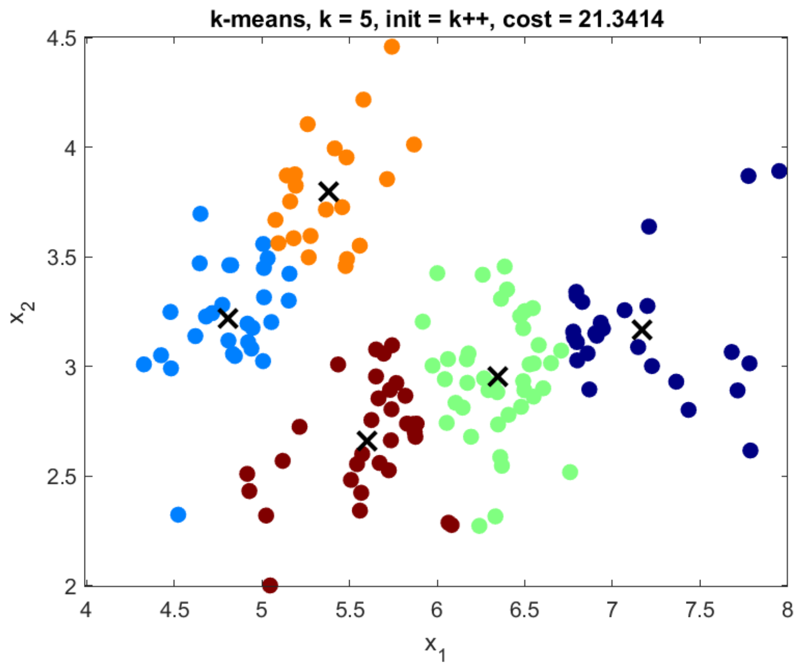
Appendix 4: Face reconstruction code

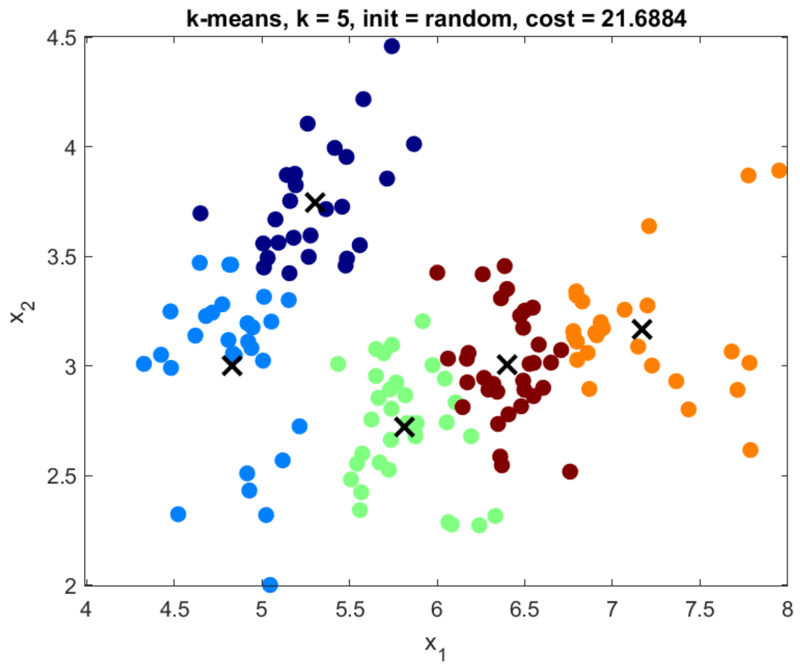
```
K = [5, 10, 50];
faceidx = [1000, 2000];

% Displaying original faces
FigHandle = figure;
imagesc(reshape(X(1000,:), [24, 24]));
colormap gray;
axis square;
title('Face 1000 Original');
saveas(FigHandle, ['Face 1000 Original' '.png']);
close all;

FigHandle = figure;
imagesc(reshape(X(2000,:), [24, 24]));
colormap gray;
axis square;
title('Face 2000 Original');
saveas(FigHandle, ['Face 2000 Original' '.png']);
close all;

% Reconstructing faces using K principal components and displaying
for k = K
    % Reconstructing all faces
    X0h = W(:, 1:k)*V(:, 1:k)';
    for f = faceidx
        % Displaying faces 1000 and 2000
        FigHandle = figure;
        imagesc(reshape(X0h(f,:), [24, 24]));
        colormap gray;
        axis square;
        titleStr = ['Face ', num2str(f), ', K=', num2str(k)];
        title(titleStr);
        saveas(FigHandle, [titleStr '.png']);
        close all;
    end
end
```

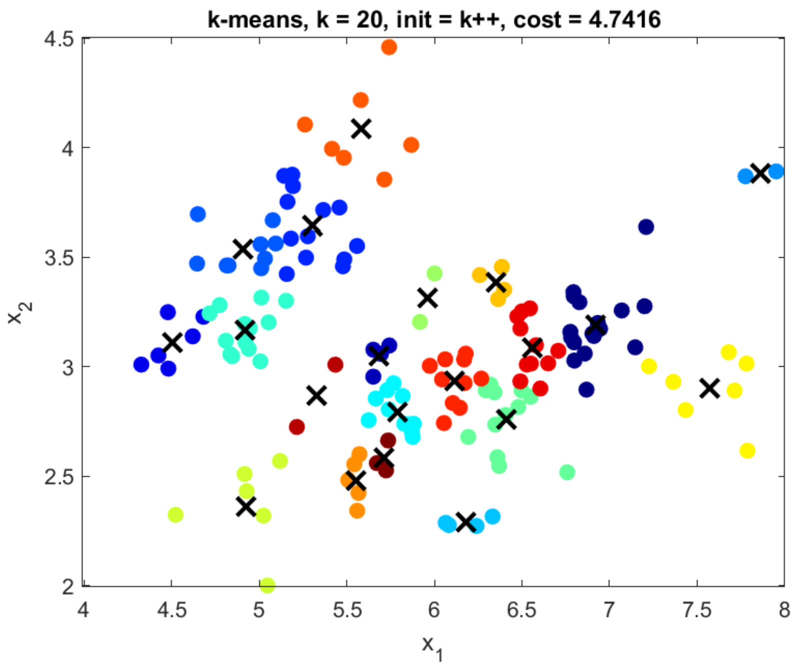
Appendix 5: Plotting first 2 features of iris data

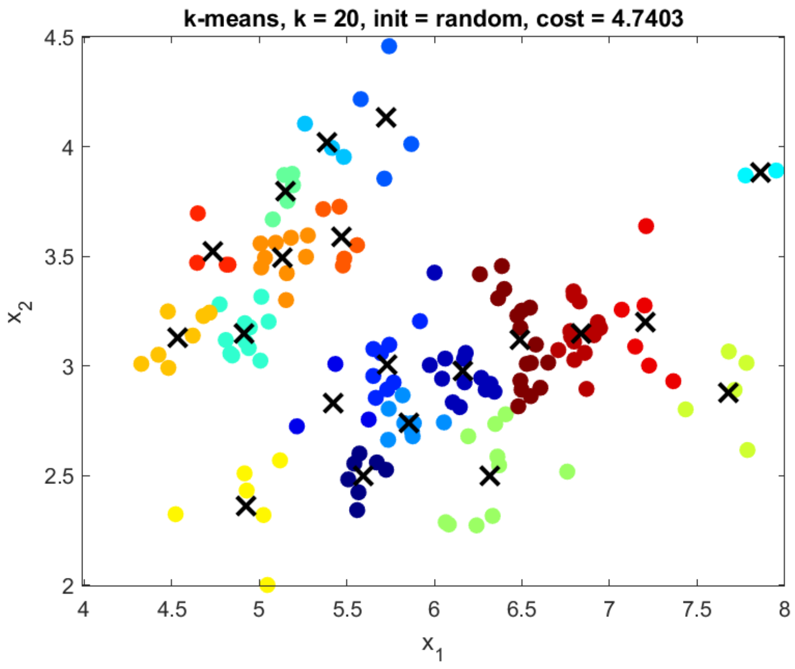
```
FigHandle = figure;
plot(data(:,1), data(:,2), '.', 'MarkerSize', 24);
title('First two features of iris data');
xlabel('x_1');
ylabel('x_2');
saveas(FigHandle, ['Iris Data' '.png']);
```

Appendix 6.1: K-Means additional initializations






k-means, k = 20, init = k++, cost = 4.7416
4.5
4
3.5
3
2.5
2
x_2
4
4.5
5
5.5
6
6.5
7
7.5
8
x_1


k-means, k = 20, init = random, cost = 4.7403
4.5
4
3.5
3
2.5
2
x_2
4
4.5
5
5.5
6
6.5
7
7.5
8
x_1

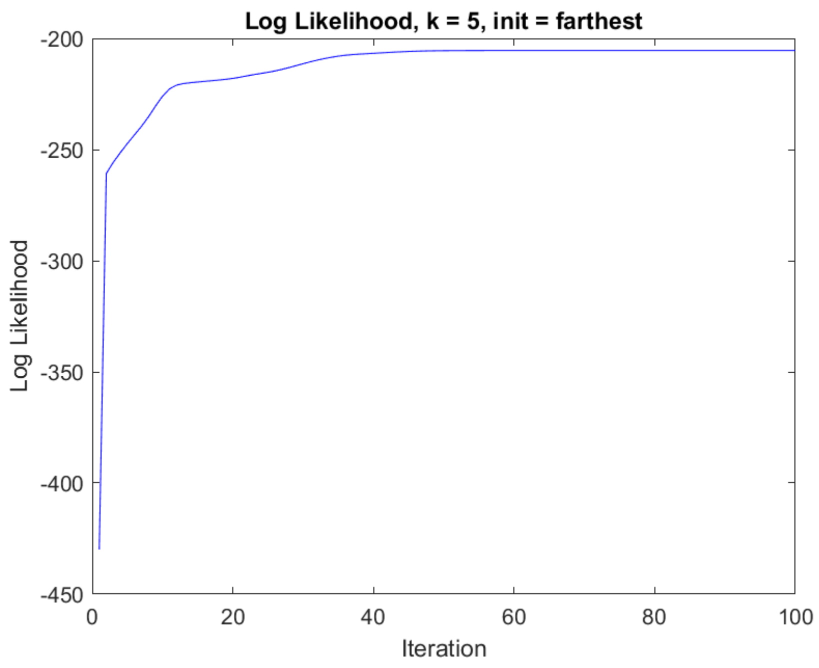
Appendix 6.2: K-Means code

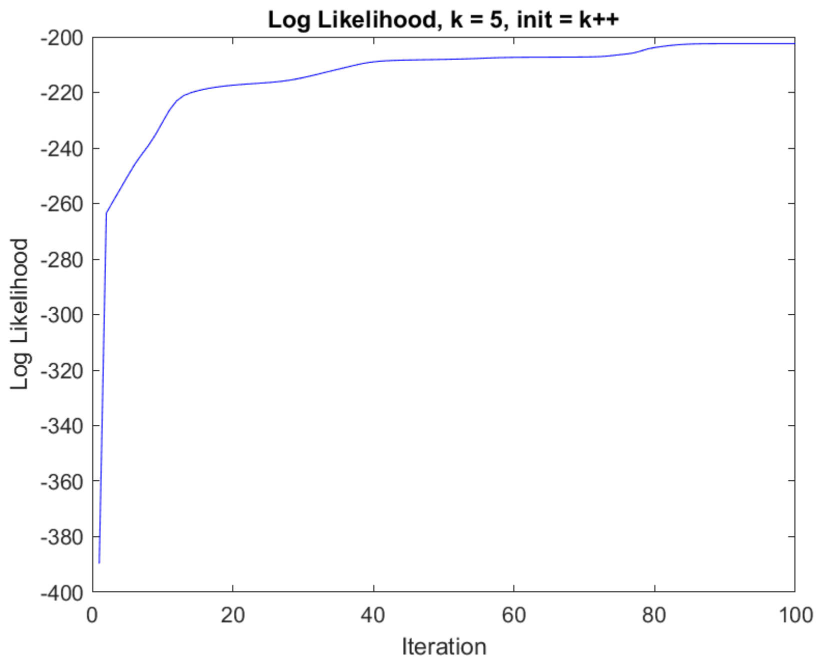
```
close all;
initialization = {'random', 'farthest', 'k++'};
for i = 1:3
    for k = [5, 20]
        [z, c, cost] = kmeans(data, k, initialization{i});

        FigHandle = figure;
        plotClassify2D([], data, z); % Plot Clusters
        hold on;
        plot(c(:,1), c(:,2), 'kx', 'MarkerSize', 12, 'LineWidth', 2); % Plot Centroids
        hold off;

        titleStr = ['k-means, k = ', num2str(k), ', init = ', initialization{i}, ', cost
= ', num2str(cost)];
        title(titleStr);
        xlabel('x_1');
        ylabel('x_2');

        saveas(FigHandle, [titleStr '.png']);
    end
end
```

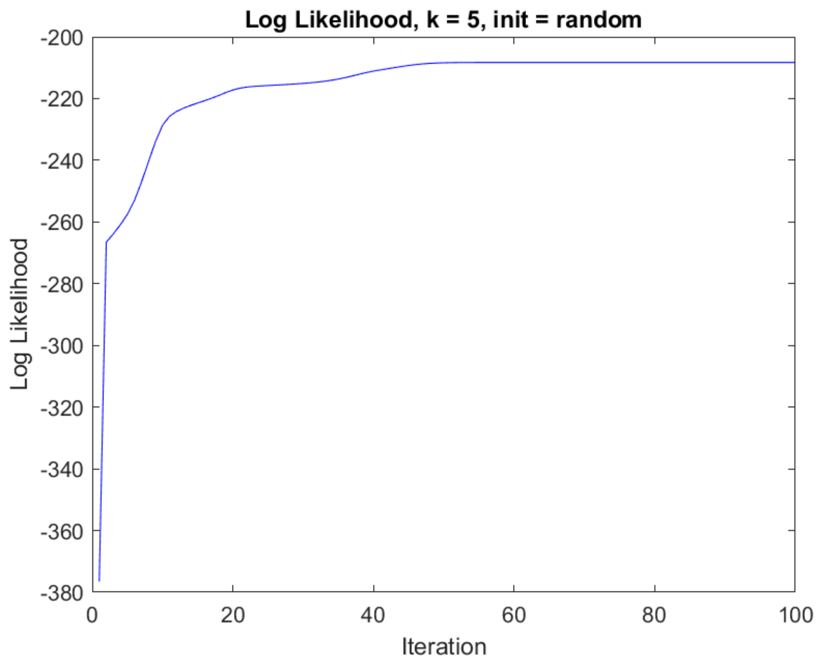
Appendix 7: Agglomerative Clustering code

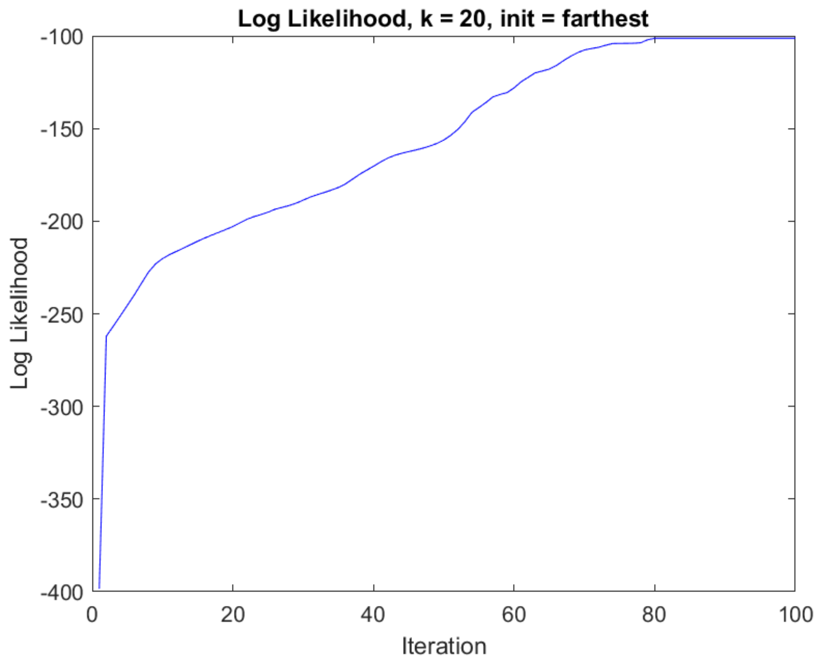
```
close all;
link = {'min', 'max'};
for i = 1:2
    for k = [5, 20]
        [z, join] = agglomCluster(data, k, link{i});

        FigHandle = figure;
        plotClassify2D([], data, z); % Plot Clusters

        titleStr = ['Agglomerative Clustering, k = ', num2str(k), ', link = ', link{i}];
        title(titleStr);
        xlabel('x_1');
        ylabel('x_2');

        saveas(FigHandle, [titleStr '.png']);
    end
end
```

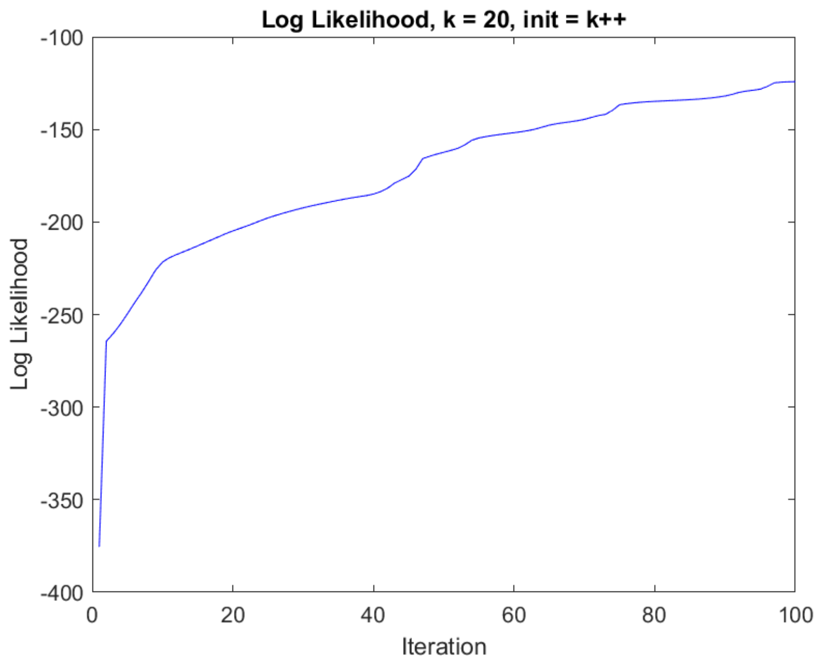
Appendix 8.1: Log likelihood maximization figures for GMM

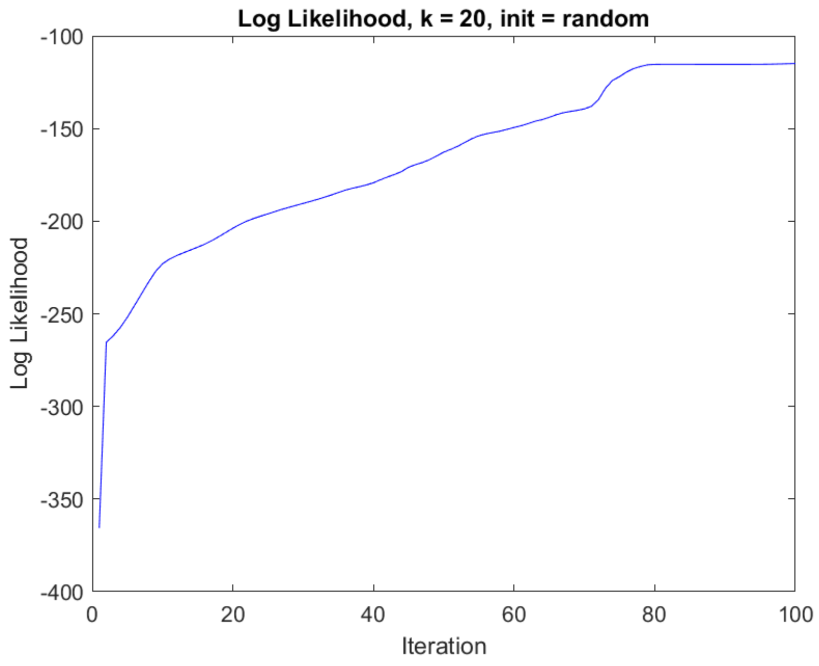





Log Likelihood, k = 5, init = random
Log Likelihood
Iteration
-200
-220
-240
-260
-280
-300
-320
-340
-360
-380
0
20
40
60
80
100


Log Likelihood, k = 20, init = farthest
Log Likelihood
Iteration
-100
-150
-200
-250
-300
-350
-400
0
20
40
60
80
100


Log Likelihood, k = 20, init = k++
Log Likelihood
Iteration
-100
-150
-200
-250
-300
-350
-400
0
20
40
60
80
100


Log Likelihood, k = 20, init = random
Log Likelihood
Iteration
-100
-150
-200
-250
-300
-350
-400
0
20
40
60
80
100

Appendix 8.2: GMM figure generation code

```
close all;
initialization = {'random', 'farthest', 'k++'};
for i = 1:3
    for k = [5, 20]
        [z,T,soft,ll] = emCluster(data, k, initialization{i}); % Running EM GMM

        % -- Code to save figures as images -- %
        FigList = findobj(allchild(0), 'flat', 'Type', 'figure');
        for iFig = 1:length(FigList)
          FigHandle = FigList(iFig);
          FigName   = ['k_', num2str(k), ' init_', initialization{i}];
          set(0, 'CurrentFigure', FigHandle);
          if(iFig == 1)
              titleStr = ['Log Likelihood, k = ', num2str(k), ', init = ',
initialization{i}];
              xlabel('Iteration');
              ylabel('Log Likelihood');
              title(titleStr);
              saveas(FigHandle, [titleStr '.png']);
          else
              titleStr = ['GMM, k = ', num2str(k), ', init = ', initialization{i}];
              xlabel('x_1');
              ylabel('x_2');
              title(titleStr)
              saveas(FigHandle, [titleStr '.png']);
          end
        end
        % -- End of figure save -- %
    end
end
```

Appendix 9: Setup

```
%% Set up
close all;
clc;
clear;

load data_ps3_2.mat
C = 1000;
```

Appendix 10: Data Set 1

```
%% Data Set 1

% Testing with Klinear
svm_test(@Klinear, 1, C, set1_train, set1_test)
% TEST RESULTS: 0.0446 of test examples were misclassified.

% Testing with Kpoly
svm_test(@Kpoly, 2, C, set1_train, set1_test)
% TEST RESULTS: 0.0514 of test examples were misclassified.

% Testing with Kgaussian
svm_test(@Kgaussian, 1, C, set1_train, set1_test)
% TEST RESULTS: 0.0571 of test examples were misclassified.

% Conclusion:
% The best kernel is the linear one as the data set clearly shows a linear
% decision boundary separating the data. In these cases, it is best to
% avoid overfitting, hence choosing the simplest model. The error rates on
% the test examples also support this claim as the linear decision boundary
% gave the lowest amounts of misclassified testing data (0.0446 of test
% examples misclassified).
```

Appendix 11: Data Set 2

```
%% Data Set 2

% Testing with Klinear
svm_test(@Klinear, 1, C, set2_train, set2_test)
% TEST RESULTS: 0.273 of test examples were misclassified.

% Testing with Kpoly
svm_test(@Kpoly, 2, C, set2_train, set2_test)
% TEST RESULTS: 0.011 of test examples were misclassified.

% Testing with Kgaussian
svm_test(@Kgaussian, 1, C, set2_train, set2_test)
% TEST RESULTS: 0.014 of test examples were misclassified.

% Conclusion:
% The best kernel is the second order polynomial one as the data set
% clearly shows a parabola of degree 2 separating the data. In this case,
% the linear kernel would result in being underfitting, whereas the
% Gaussian kernel would overfit. The error rates on the test examples also
% support this claim as the polynomial kernel of degree 2 gave the lowest
% amounts of misclassified testing data (0.011 of test examples were
% misclassified).
```

Appendix 12: Data Set 3

```
%% Data Set 3

% Testing with Klinear
svm_test(@Klinear, 1, C, set3_train, set3_test)
% TEST RESULTS: 0.471 of test examples were misclassified.

% Testing with Kpoly
svm_test(@Kpoly, 2, C, set3_train, set3_test)
% TEST RESULTS: 0.132 of test examples were misclassified.

% Testing with Kgaussian
svm_test(@Kgaussian, 1, C, set3_train, set3_test)
% TEST RESULTS: 0 of test examples were misclassified.

% Conclusion:
% The best kernel is the Gaussian one as the data set clearly shows groups
% of data points clustered, making it not separable with the linear or
% polynomial kernel. In this case, the Gaussian kernel is able to separate
% these groups, hence it is the best choice. The error rates on the test
% examples also support this claim as the Gaussian kernel produced no
% amounts of misclassified testing data.
```

Appendix 13: SVM Problem 2 - Data Set 4

```
%% SVM: Problem 2

% Testing with Klinear
svm_test_digital(@Klinear, 1, C, set4_train, set4_test)
% TEST RESULTS: 0.14 of test examples were misclassified.

% Testing with Kpoly
svm_test_digital(@Kpoly, 2, C, set4_train, set4_test)
% TEST RESULTS: 0.12 of test examples were misclassified.

% Testing with Kgaussian
svm_test_digital(@Kgaussian, 1.5, C, set4_train, set4_test)
% TEST RESULTS: 0.085 of test examples were misclassified.

% Conclusion:
% The linear kernel provided results of 0.14 misclassified test examples.
% The second order polynomial kernel provided results of 0.12 misclassified
% test examples.
% The Gaussian kernel provided results of 0.085 misclassified test
% examples.
```

Appendix 14: svm_test_digital.m code

```
function [] = svm_test_digital(kernel,param,C,train_data,test_data)

% Get the SVM
svm = svm_train(train_data, kernel, param, C);

% Verify for training data
y_est = sign(svm_discrim_func(train_data.X, svm));
error = find(y_est ~= train_data.y);

if (error)
    fprintf('WARNING: %d training examples were misclassified!!!\n',length(error));
end

% Evaluate against test data
y_est = sign(svm_discrim_func(test_data.X, svm));
error = find(y_est ~= test_data.y);

fprintf('TEST RESULTS: %g of test examples were misclassified.\n',...
    length(error)/length(test_data.y));
```