University of New South Wales

School of Electrical Engineering and Telecommunications

INDIVIDUAL Assessment Task: Part 2 Assignment 1

Course Code	ELEC9782	Course Name	Data science
Week/Session/Year	Year 4	Lecturer	Dr. Vidhya

Student Number	z5036602
Family Name	Liu
Given Names	Zhengyue

Mark/Grade given (For official use only)

Marker's Comments:

Since this work counts toward your formal assessment for this course, please write your name and student number where indicated above, and sign the declaration below. Attach this cover sheet to the front of your submission, so that your name and student number can be seen without any cover needing to be opened.

For further information, please see http://www.lc.unsw.edu.au/plagiarism/index.html

I declare that this assessment item is my own work, except where acknowledged, and has not been submitted for academic credit elsewhere, and acknowledge that the assessor of this item may, for the purpose of assessing this item:

- ***Reproduce this assessment item and provide a copy to another member of the University ; and/or,***
- ***Communicate a copy of this assessment item to a plagiarism checking service (which may then retain a copy of the assessment item on its database for the purpose of future plagiarism checking).***

I certify that I have read and understood the University Rules in respect of Student Academic Misconduct.

Signature of student ____________ Date: 10/08/2019

ELEC9782 Assignment 1

z5036602 - Zhengyue LIU

Semester 2 2019

I, Zhengyue LIU(student number z5036602), declare that the following assignment is my own work and that I have read and understood the University Rules in respect of Student Academic Misconduct.

Question 1

Problem (a)

The distance from multivariate normal distribution of a k-dimensional random vector $\mathbf{X} = (\mathbf{X}_1, ..., \mathbf{X}_k)^T$, $\mathbf{X_i} \sim \mathbf{N}(0, \sigma^2)$ to mean vector $\mathbf{U} = (u_1, u_2, ..., u_k)^T$ can be written as:

$$\begin{aligned} D &= \sqrt{(\mathbf{X}_1 - u_1)^2 + (\mathbf{X}_2 - u_2)^2 + ... + (\mathbf{X}_k - u_k)^2} \\ &= \sigma \sqrt{\frac{(\mathbf{X}_1 - u_1)^2}{\sigma^2} + \frac{(\mathbf{X}_2 - u_2)^2}{\sigma^2} + ... + \frac{(\mathbf{X}_k - u_k)^2}{\sigma^2}} \end{aligned}$$

$\sum_k (\frac{\mathbf{X}_k - u_k}{\sigma})^2$ is a standardized chi-squared distribution of degree k. Let Z be $\sum_k (\frac{\mathbf{X}_k - u_k}{\sigma})^2$, then $D = \sigma\sqrt{Z}$. Cumulative distribution for D is:

$$\mathbf{F}_D(d) = \mathbf{P}(D \leq d) = \mathbf{P}(\sigma\sqrt{Z} < d) = \mathbf{P}(Z < \frac{d^2}{\sigma^2}) = \mathbf{F}_Z(\frac{d^2}{\sigma^2})$$

$\mathbf{F}_z$ is the cdf of chi-squared distribution of degree k(dimension).

$$f_D(d) = \mathbf{F}'_D(d) = \frac{2d}{\sigma^2}\mathbf{F}'_Z(\frac{d^2}{\sigma^2}) = \frac{2d}{\sigma^2} f_Z(\frac{d^2}{\sigma^2})$$

f_Z is the pdf of chi-squared distribution of degree k.

PDF plots

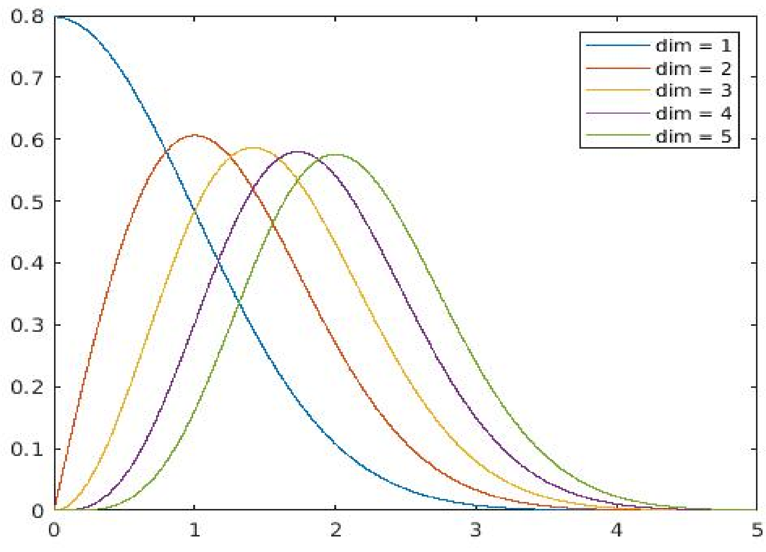


Figure 1: pdf of distance between data and mean for 1 dimension to 5 dimensions

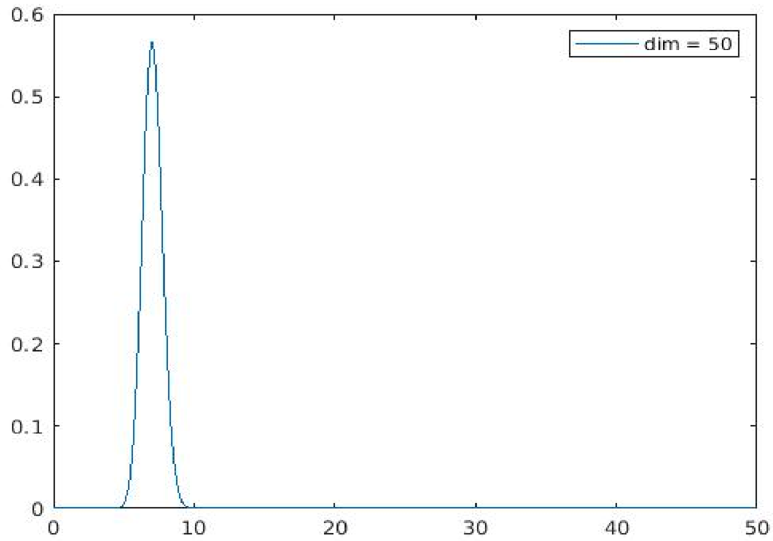


Figure 2: pdf of distance between data and mean for 50 dimensions

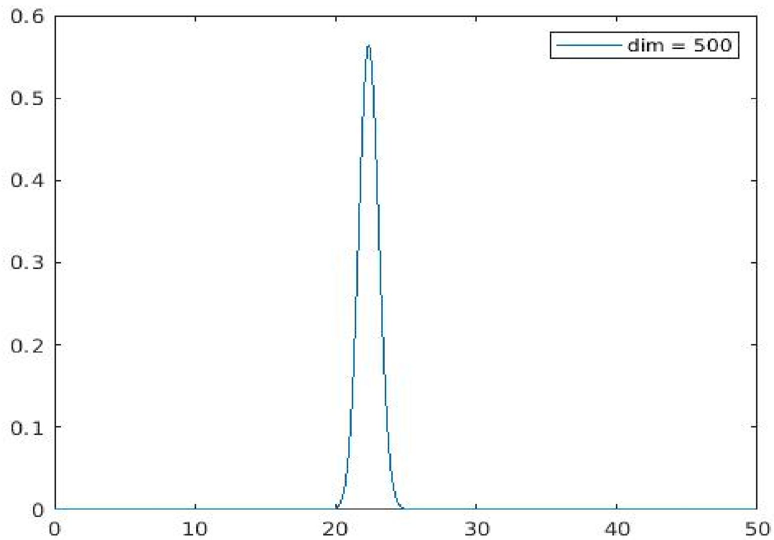


Figure 3: pdf of distance between data and mean for 500 dimensions

Data in High dimensional space tends to be sparser than in lower dimensions. Data tends to lie further from the origin as the dimension increased. They mostly accumulate in a thin hollow sphere. As the dimension increased, the radius of the sphere increases. This phenomenon can be observed from above figures. As the dimension increases, the pdf function shifts to the right and creates a narrow probability "band".

Matlab implementation

The `pdf_plot.m` is used. `chi2pdf` built-in function is used to generate the chi-squared distribution. The likelihood over the different distance is calculated by using the pdf derived above. Finally, the likelihood is plotted for different dimensions.

Problem (b)

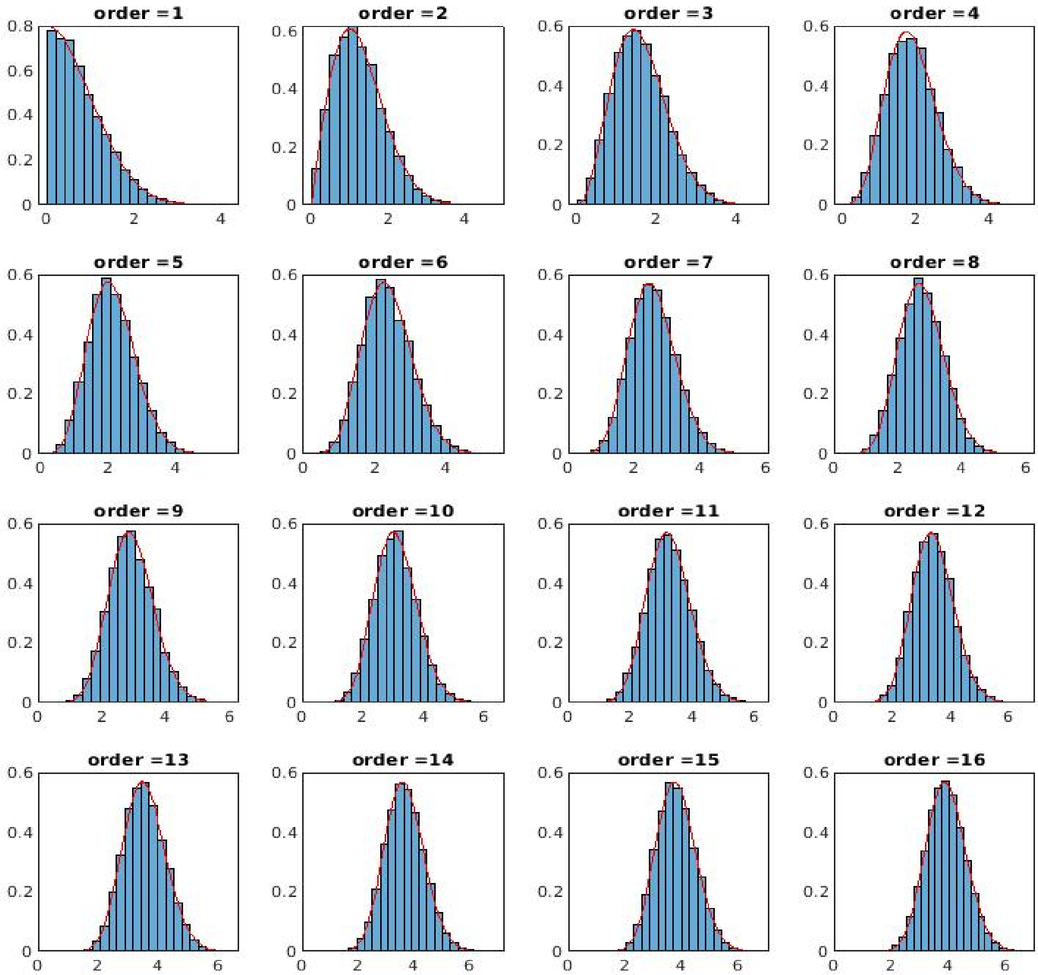


Figure 4: empirical pdf estimation by using histogram for 1 dimension to 16 dimensions, the red line is the theoretical pdf, blue bins are histogram bins. The empirical estimation agrees on theoretical values

Matlab implementation

The `empirical_estimates.m` is used. Firstly, multivariate guassian distributed data are generated using `randn` built-in function. The dimension of data range from 1 to 16. The distance is calculated using `norm` built-in function. Finally, the histogram of the distances is plotted for different dimensions. The theoretical pdf is also plotted on same graph for comparison. The code has been commented and attached in Q1 section in Appendix.

Problem (c)

Cosine distance is a good measure of similarity/dissimilarity. Cosine distance calculates the angle between two data vectors, and use it as a measure of similarity. Smaller angle means two data vector are more similar to each other. In high-dimensional space, the curse of high dimensionality and sparsity will cause Euclidean distance based methods to fail. So, the angle measure between two vectors is a second best choice.
Another way to think about it is that as data mainly lies in a thin hollow sphere, and the distance between different data and the origin are similar, we can approximate such a thin hollow sphere as a spherical surface and all the data lies on this surface. The distance between different data and the origin can be considered as the radius r of the spherical surface. Then, the arc length between two data points on this surface is calculated as:

$$\text{arc length} = \theta * r$$

Hence, smaller theta means two data points are closer to each other in high-dimensional space. Whereas higher theta will give opposite result.

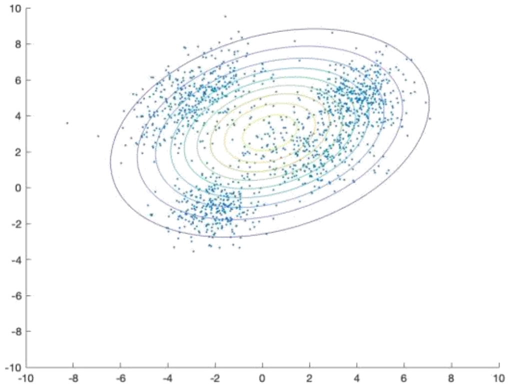
Question 2

Problem (a)

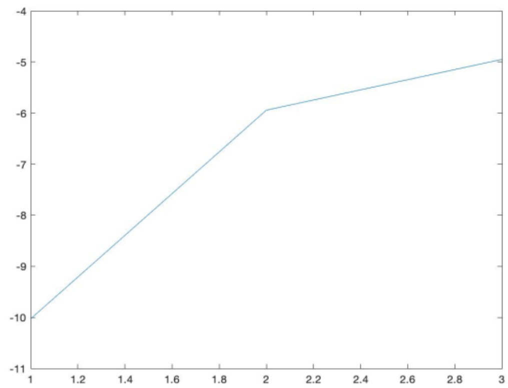
i. Performance testing by visualisation (2-D)

Testing data are generated by 2-D Gaussian mixture with 4 components:

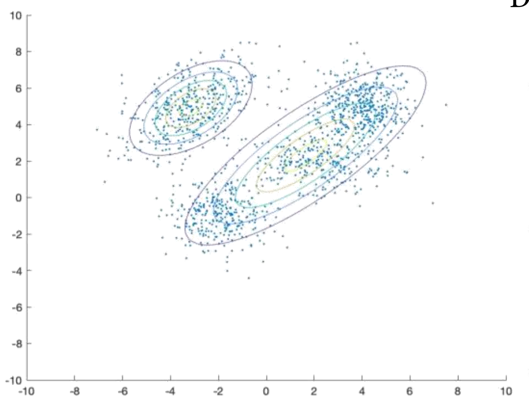
A



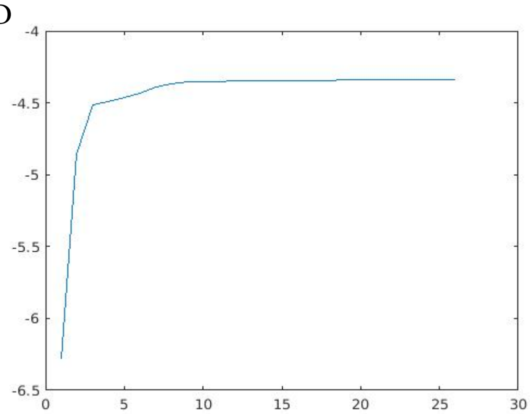
B



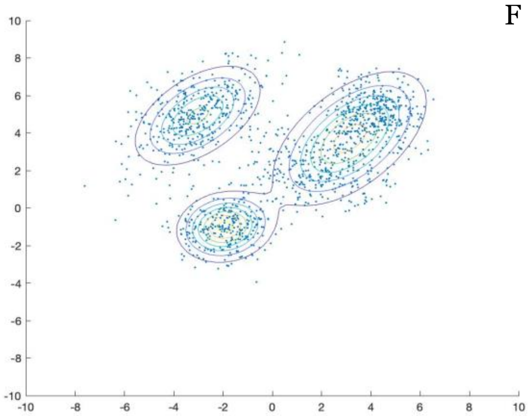
C



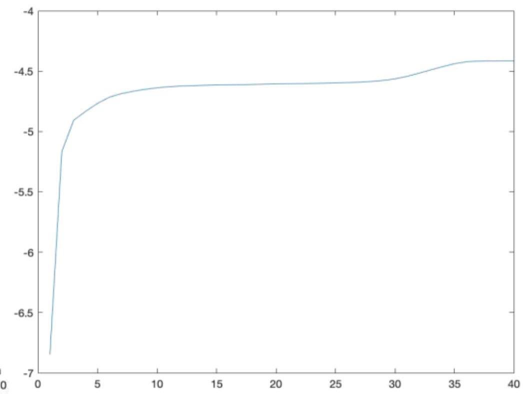
D



E



F



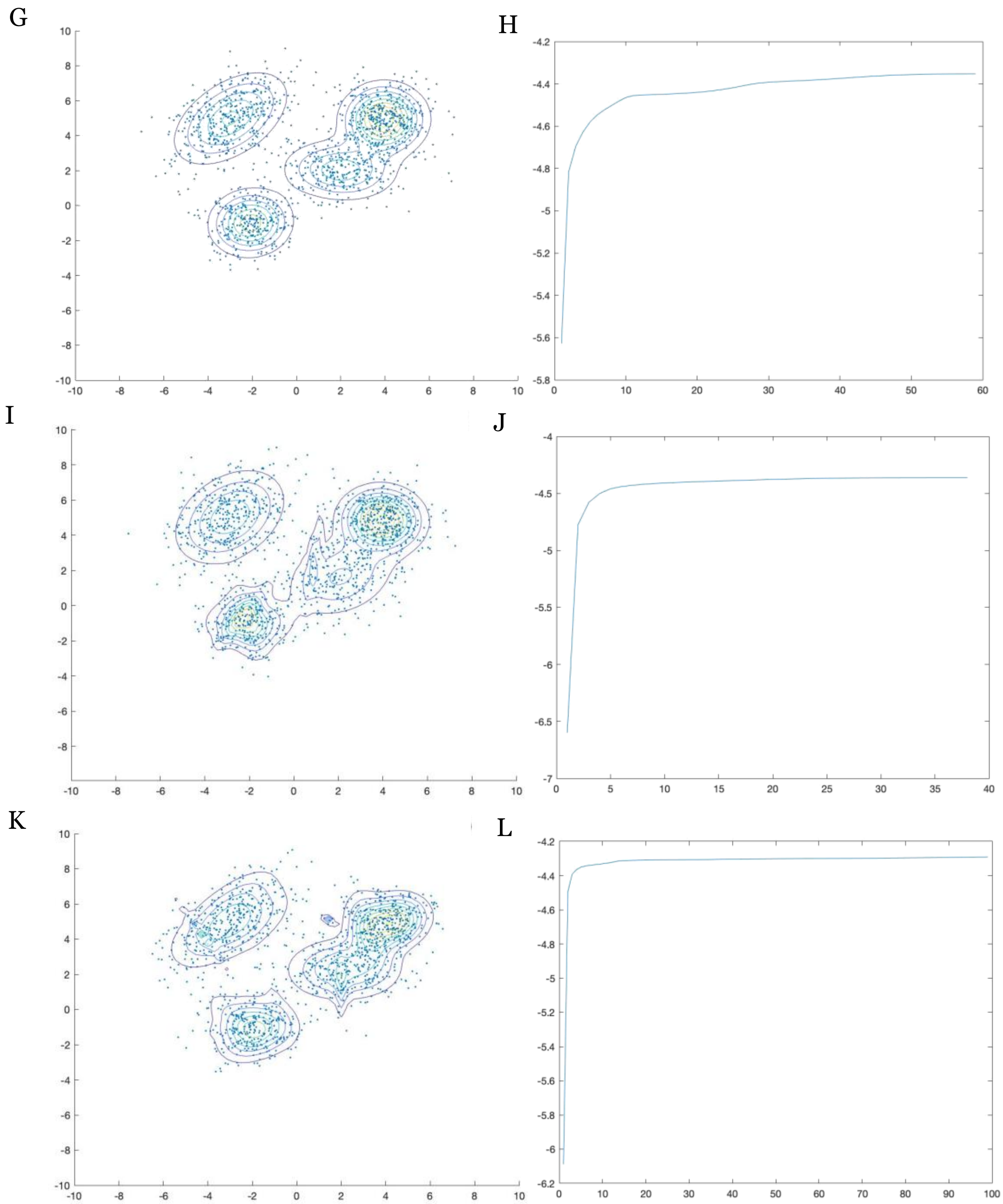


Figure 1. Contour plots of fitted Gaussian mixture (left) and plots of log-likelihood of data given model v.s. EM iterations (right) when **A)** and **B)** one, **C)** and **D)** two, **E)** and **F)** three, **G)** and **H)** four, **I)** and **J)** seven, and **K)** and **L)** fourteen components were fitted.

As we can see, when we try to fit Gaussian mixture model with the number of components larger than that of the true Gaussian mixture model, the effect of extra components is negligible in contour plot. We can conclude that there is an optimal number of components for fitting a Gaussian mixture. Fitting too many components can easily introduce singularity problem and cause inaccuracy and program break-down. The way I solved the singularity problem is introduced in Matlab implementation section.

ii. Performance testing Predicted parameters v.s. True parameters

Testing data generated by a 3-D GMM with 3 components:

	μ_1			μ_2			μ_3		
True μ	2	2	2	-2	-1	-1	4	5	6
Predicted μ	2.1	1.97	2.01	-1.99	-0.98	-1.02	4.02	4.97	6.00
	Σ_1			Σ_2			Σ_3		
True Σ	$\begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix}$			$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$			$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$		
Predicted Σ	$\begin{pmatrix} 1.90 & -0.01 & 0.01 \\ -0.01 & 2.20 & 0.06 \\ 0.01 & 0.06 & 2.20 \end{pmatrix}$			$\begin{pmatrix} 0.93 & -0.03 & 0.01 \\ -0.03 & 0.95 & 0.03 \\ 0.01 & 0.03 & 0.92 \end{pmatrix}$			$\begin{pmatrix} 0.97 & -0.05 & 0.07 \\ -0.05 & 0.89 & 0.01 \\ 0.07 & 0.01 & 1.01 \end{pmatrix}$		

Testing data generated by a 4-D GMM with 3 components:

	μ_1				μ_2				μ_3			
True μ	3	2	1	5	-2	-1	-1	0	4	5	6	7
Predicted μ	2.99	1.97	0.74	4.87	-2.01	-0.98	-1.10	-0.02	4.14	5.03	5.92	6.86
	Σ_1				Σ_2				Σ_3			
True Σ	$\begin{pmatrix} 4 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 4 \end{pmatrix}$				$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$				$\begin{pmatrix} 3 & 2 & 0 & 0 \\ 2 & 3 & 0 & 0 \\ 0 & 0 & 3 & 1 \\ 0 & 0 & 1 & 3 \end{pmatrix}$			
Predicted Σ	$\begin{pmatrix} 3.74 & -0.61 & 0.26 & -0.07 \\ -0.61 & 4.11 & 0.10 & 0.12 \\ 0.26 & 0.10 & 4.02 & -0.42 \\ -0.07 & 0.12 & -0.42 & 3.85 \end{pmatrix}$				$\begin{pmatrix} 0.89 & 0.01 & 0.03 & -0.07 \\ 0.01 & 1.01 & 0.06 & -0.02 \\ 0.03 & 0.06 & 1.11 & 0.04 \\ -0.07 & -0.02 & 0.04 & 1.01 \end{pmatrix}$				$\begin{pmatrix} 2.90 & 2.03 & 0.16 & 0.19 \\ 2.02 & 3.36 & 0.13 & 0.31 \\ 0.16 & 0.13 & 3.14 & 1.24 \\ 0.19 & 0.31 & 1.24 & 3.31 \end{pmatrix}$			

iii. Matlab implementation:

The self-implemented EM algorithm follows the procedure taught in lecture. (https://subjects.ee.unsw.edu.au/elec9782/elec9782-B-lec4.pdf p.15). The code has been thoroughly commented and attached in section Q2 in Appendix. As the EM is a stochastic process, it sometimes will minimise the covariance to zero. So, it can maximise the likelihood. In other words, it may fit an infinite high probability Gaussian on a single data with zero covariance. This is called singularity problem. My code solves this problem by adding a little bit of noise into the covariance matrix, if the covariance is considered as zero by Matlab, this noise will be added to the covariance matrix to prevent the singularity. At the same time, my program will print the warning "Hit the singularity, noise added for calculation. Try to fit the data with less Gaussian and run multiple times."

List of build-in functions used:

max()	diag()	sum()
min()	sqrt()	fprintf()
zeros()	det()	eye()
ones()	error()	rank()
length()	exp()	size()
plot()	isscalar()	ismatrix()
asset()	iscolumn()	

Question 3

Problem (a)

i. Prediction Result:

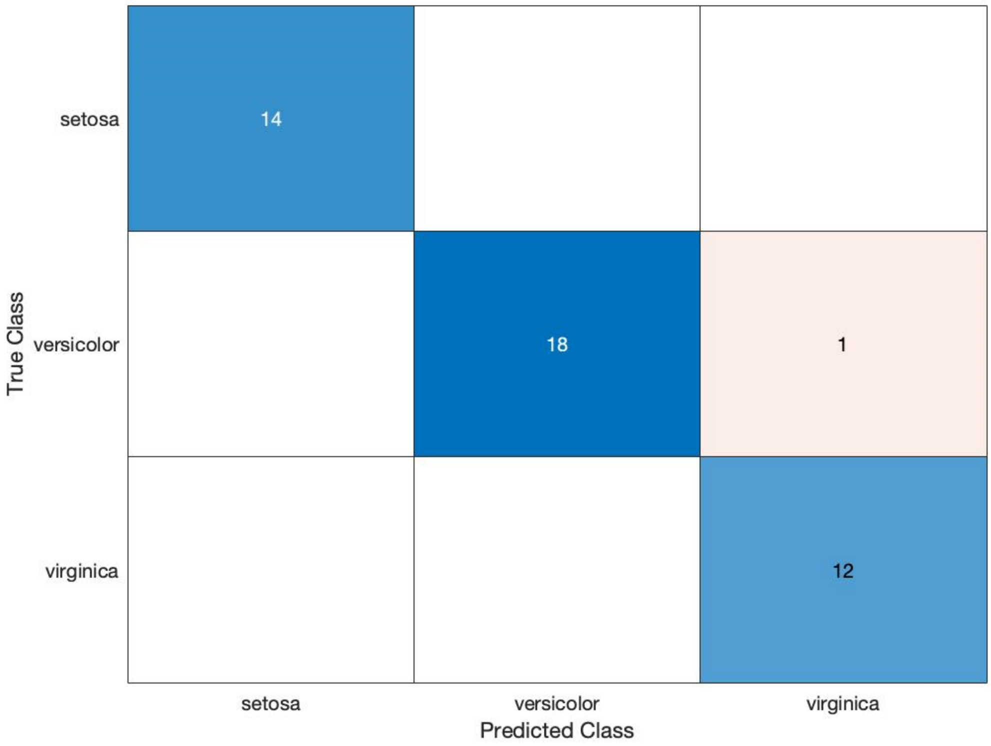


Figure 2. Confusion matrix of the prediction results.

Error rate: 2.22%

ii. Matlab implementation:

I used logistic regression (one v.s. all) model for the classification. The data are shuffled and splitted into the training and the test set. I change the multi-class classification problem to the binary classification problem for each class. The data belongs to the chosen class will be labeled as 1(true). The data outside of the chosen class will be labeled as 0 (false) . Then, we train a logistic regression model on the chosen class by gradient descent. Each class will have their own logistic regression model.
When a new data comes in for class-label predication. We will calculate the probability score of data under each class's logistic regression model. Finally, we pick the class label which has the highest probability score as the final prediction.

The logistic regression is easy for implementation and often have promising results for supervised learning. As we have only three class labels, training three logistic regression models is not too costly.

The code has been thoroughly commented and attached in section Q3 in Appendix.

List of build-in functions used:

mean()	unique()	confusionchart()
std()	find()	
disp()	ones()	
num2str()	str2double()	
zeros()	sum()	
size()	ceil()	
nemul()	length()	

Appendix

Q1

pdf_plot.m

```
clc;clear;
sigma = 1;
D = [0:5/1000001:5];
figure;
chi_index = (D/sigma).^2;
%%%caculate the pdf and plot the function
for order = 1:5
    chi_distribution = chi2pdf(chi_index,order);
    f_D = ((2*D)/sigma^2).*chi_distribution;
    plot(D,f_D,'DisplayName', [ 'dim = ' num2str(order)]);
    hold on
end
legend('show'); %create/show legend
hold off
```

empirical_estimates.m

```
%%%generate the histogram plot for distance pdf from dim 1 to k
n = 1e4;
k = 4^2;
figure
for j = 1:k
    subplot(sqrt(k),sqrt(k),j)
    %generate multivariate data
    x = randn(j,n);
    d = zeros(1,n);
    %caculate and store the norm(distance) of the data
    for i = 1:n
        d(i) = norm(x(:,i));
    end
    %plot the histogram of the distances
    histogram(d,20,'Normalization','pdf')
    hold on

    %%%plot theoretical pdf
    sigma = 1;
    pdfd = @(d) 2.*d./(sigma^2).*chi2pdf(d.^2/(sigma^2),j);
    fplot(pdfd,[0,max(d)],'r');
    title(['order =' num2str(j)]);
    hold off
end
```

Q2

EM.m

```
%%%implementation of EM algorithm
%input:
%X --> data with no restriction on dimension and observations
%init --> numebr of gaussians we want
%output:
%gmm_model--> a struct stores number of components and paramters of model
function [gmm_model] = EM(X,init)
    %%%initialisation process
    %initilise number of componentes
    gmm_model.num_of_components = init;

    %randomly initilise gaussian means. We get maximum and minimum value
    %along all dimensions of all value, and randomly choose value within this
    %range as the value along each dimension of our mean.
    maximum_X = max(X,[],'all');
    minimum_X = min(X,[],'all');
    r = minimum_X+(maximum_X-minimum_X).*rand(init,size(X,2));
    gmm_model.u = r;
    %%% initialise the covariance matrix by a constant times identity
    %%% matrix. This constant is caculated as difference between
    %maximum and minimum value divided by
    %number of components.
    sigma = zeros(size(X,2),size(X,2),init);
    for k = 1:init
        sigma(:,:,k) = ((maximum_X-minimum_X)/init)*eye(size(X,2));
    end
    gmm_model.sigma = sigma;
    %%%initialise the weights,we assmue all weights are equal
    weights = (1/init)*ones(1,init);
    gmm_model.weights = weights;

    %%%EM process
    iteration = 500;                %%maximum num of iterations
    prev_llh_vec = [];              %%dynamic array for storing the log-liklihood of ...
        data after each iteration
    similarity_hitted = 0;          %%flag for checking if EM hitted singularity
    for it = 1:iteration
        %%%update the parameter
        [gmm_model,prev_llh,singularity_flag] = updation(X,gmm_model);
        %%%check if EM hitted the singularity
        if singularity_flag == 1
            similarity_hitted =1;
        end
        %%%break at log-liklihood become steady
         if it ≠ 1 && prev_llh-prev_llh_vec(end)<1e-4
             break
         end
        %%%store the log-liklihood to show it is maximised
        prev_llh_vec = [prev_llh_vec prev_llh];

    end

    %%%report the similarity problem
    if similarity_hitted ==1
        fprintf(['Hitted the singularity,noise added for caculation \n Try to fit the ...
            ', ...
        'data with less gaussians,and run multiple times\n']);

    end
    %%%plot log-lilihood change during the EM process.
    plot(1:length(prev_llh_vec),prev_llh_vec);



end
```

updation.m

```matlab
%%%this function update the parameters by expectation maximisation
%   when EM hit the singularity, similar to sklearn,small noise $1e-6 * Identity matrix$
%   will be added with covariance matrix to prevent singularity.
%   the formula used here is from ...
    https://subjects.ee.unsw.edu.au/elec9782/elec9782-B-lec4.pdf
%   slide 15.
%input:
%   X -- > data
%   gmm_model_old-->a struct stores original model parameters
%output:
%   gmm_model_new-->a struct stores new model parameters
%   singularity_flag --> report whether the EM hit the singularity point

function [gmm_model_new,llh,singularity_flag] = updation(X,gmm_model_old)
    %%%extract the parameters from model struct and assert the input correctness
    u_old = gmm_model_old.u;
    num_of_components = gmm_model_old.num_of_components;
    assert(size(u_old,1)==num_of_components,'number of means does not agree on num of ...
        components');
    assert(size(u_old,2)==size(X,2),'means dimension does not agree on num of components');
    %%%caculate the posterior probability gamma given data.
    num_of_observations = size(X,1);
    singularity_flag = 0;
    [gamma_matrix,llh] = latent_pospdf(X,gmm_model_old);

    %%%update the paramters
    %%%https://subjects.ee.unsw.edu.au/elec9782/elec9782-B-lec4.pdf
    %%%page 15.
    N_k = sum(gamma_matrix,1);

    u_new = (gamma_matrix'*X)./N_k';
    sigma_new = zeros(size(X,2),size(X,2),num_of_components);
    for m = 1:num_of_components
        sigma_sum=0;
        for n = 1:num_of_observations
            U = (X(n,:)-u_old(m,:))'*(X(n,:)-u_old(m,:));
            sigma_sum=sigma_sum+gamma_matrix(n,m)*U;

        end
        sigma_new(:,:,m) =sigma_sum/N_k(m);

        %%%prevent the singularity,similar to sklearn approach.
        if rank(sigma_new(:,:,m)) < size(sigma_new(:,:,m),2)
             sigma_new(:,:,m)=sigma_new(:,:,m)+1e-6*eye(size(sigma_new(:,:,m),2));
             singularity_flag = 1;
        end

    end
    weights = N_k/num_of_observations;

    %%%construct the new model struct with updated parameters
    gmm_model_new.u = u_new;
    gmm_model_new.sigma = sigma_new;
    gmm_model_new.weights = weights;
    gmm_model_new.num_of_components = gmm_model_old.num_of_components;
end
```

latent_pospdf.m

```
%   Given the model and a data point,this function can caculate the posterior
%   pabability of p(z|x u sigma) for ***all the observations***.

%   p(z|x u sigma)= pi_k*N(x|theta_k)/ sum_{j}(pi_j*N(x|theta_j)).
%   it gives us for each datapoint x_i the measure of:
%   "prabability that a given data belongs to a certain class/probability of x_i over ...
    all classes"
%   this function is similar to matlab built-in function "posterior()"

%   For convenience this function also ouputs the total log-likelihood wich is
%   essentially the log of sum_{j}(pi_j*N(x|theta_j))(the mariginal
%   likelihood of the X)
%input: x-->data (no restrictions on obervtions and dimensions)
%       gmm_model-->a struct stores number of components and parameters of the model
%
%output:gamma-->posteriror probability matrix, the row represents
%               observations,cololumn corresponds to probability belongs to
%               each gaussian distribution.
%       llh --> total log liklihood of data given current updated model



function [gamma,llh] = latent_pospdf(x,gmm_model)
    %%%extract the value from model struct and assert the input correctness
    u = gmm_model.u;
    sigma = gmm_model.sigma;
    weights = gmm_model.weights;
    num_of_components = gmm_model.num_of_components;
    assert(isvector(weights),'models weights input are wrong');
    assert(num_of_components==length(weights),'components number not equal wights number');
    assert(ismatrix(sigma(:,:,1)),'sigma dimension is wrong');

    %%%caculates the pi_k*N(x|theta_k),
    %the caculation here is matrixlised.Given the data matrix, row
    %represents each observation, columns represents dimension.
    %The caculated output has row represents each observation,
    %columns represents different gaussian clusters .
    num_of_observations = size(x,1);
    N = zeros(num_of_observations,num_of_components);
    for m = 1:num_of_components
         N(:,m) = pdfmvn(x,u(m,:),sigma(:,:,m));
    end
    N_pi = N*diag(weights);

    NUM = N_pi;

    %%%caculates sum_{j}(pi_j*N(x|theta_j)).
    DEN = sum(N_pi,2);

    %%%divide each data observation(rows) by denominator.
    gamma = NUM./DEN;

    %%%caculates the log-likelihood
    llh = sum(log(DEN))/num_of_observations;

end
```

pdfmvn.m

```
%this function caculates the liklihood of data by using the pdf of 1-d gaussian
%or multivariate gaussian, the function is similar to built-in function
%mvnpdf.m
%input: X-->data no restricitons on rows and columns
%       u-->mean vector of gaussian
%       sigma-->covariance-matrix of gaussian
%output: y --> probability
function y = pdfmvn(X,mu,sigma)
    %%%1-d
    if (iscolumn(X))
        assert(isscalar(mu),'u is not a scalar');
        assert(isscalar(sigma),'sigma is not a scalar');
        constant = 1/(sigma*sqrt(2*pi));
        X = X-mu;
        y = constant.*exp(-X.^2/(2*sigma^2));
    %%%2-d.For multiple data, row>1, we pick the diagonal of y as output
    elseif(ismatrix(X))
        dim = size(X,2);
        assert(size(mu,2)== dim,'u dim is not right');
        assert(size(sigma,2)== dim,'sigma dim is not right');
        assert(size(sigma,1)== dim,'sigma dim is not right');
        constant = 1/sqrt(det(sigma)*((2*pi)^dim));
        X = X-mu;
        U = X/sigma;
        z = (-1/2)*(U*X');
        y = diag(constant.*exp(z));

    else

        error('wrong input for pdfmvn');

    end

end
```

test.m

```
clear;clc
mu1 = [3 2 1 5];            % Mean of the 1st component
sigma1 = [4 0 0 0; 0 4 0 0; 0 0 4 0; 0 0 0 4];  % Covariance of the 1st component
mu2 = [-2 -1,-1, 0];         % Mean of the 2nd component
sigma2 = [1 0 0 0; 0 1 0 0; 0 0 1 0; 0 0 0 1];  % Covariance of the 2nd component
mu3 = [4 5 6 7];            % Mean of the 3rd component
sigma3 = [3 2 0 0; 2 3 0 0; 0 0 3 1; 0 0 1 3];  % Covariance of the 2nd component
mu4 = [-3 5 6 4];           % Mean of the 4th component
sigma4 = [2 1 1 1; 1 2 1 1; 1 1 2 1; 1 1 1 1];  % Covariance of the 2nd component

rng('shuffle')                  % For reproducibility
r1 = mvnrnd(mu1,sigma1,300);
r2 = mvnrnd(mu2,sigma2,300);
r3 = mvnrnd(mu3,sigma3,400);
%r4 = mvnrnd(mu4,sigma4,400);
X = [r1; r2;r3];

[gmm_model] = EM(X,3);

figure;
scatter(X(:,1),X(:,2),10,'.')    % Scatter plot with points of size 10
hold on
gm = gmdistribution(gmm_model.u,gmm_model.sigma,gmm_model.weights);
gmPDF = @(x,y)reshape(pdf(gm,[x(:) y(:)]),size(x));
fcontour(gmPDF,[-10 10 -10 10]);
title('Contour plot of fitted gaussian mixture on data(14 components)')
```

Q3

main.m

```
clear;clc;
load fisheriris.mat
rng = ('default');
%%%split data to test data and train data
test_ratio = 0.3;

data_with_label = [string(species),meas];
categories = unique(string(species));
[test,train] = split_train_test(data_with_label,test_ratio);

%%%train parameter for each category(one vs all) and stored in a matrix
theta = zeros(length(categories),size(meas,2)+1)';
J = zeros(3,1);

[theta(:,1),J(1)] = model_creation(train,categories(1),0.1,10000);    %Question:
[theta(:,2),J(2)] = model_creation(train,categories(2),0.2,10000);    %quicker learning ...
    rate needed
[theta(:,3),J(3)] = model_creation(train,categories(3),0.2,10000);     %why they did ...
    not converge to same point?

%%%prediction
result = predict(theta,test);
predicted_label = categories(result);
true_label = test(:,1);

%%%error and confusion matrix
error = err_caculation (true_label,predicted_label);
disp(['Err is : ', num2str(error) , '%']);
figure
cm = confusionchart(true_label,predicted_label);
```

err_calculation.m

```
%Caculate the error by number of unmatch between ground-truth and
%prediction
%input : ture_label --> true_label
%output : predicted_label -->predicted_label
%err --> error in percent
function err = err_caculation (true_label,predicted_label)
    counter = 0;
    for k = 1: length(true_label)
        if true_label(k) ≠ predicted_label (k)
            counter = counter + 1;
        end

    end

    err = counter / length (true_label)*100;

end
```

K_shuffle.m

```
%%%The function will random shuffle positions of value in a vector
%For, Details see Knuth shuffle
%https://en.wikipedia.org/wiki/Fisher%E2%80%93Yates_shuffle
%input X--> numeric array
%output X--> shuffled numeric array
function [X] = K_shuffle(X)
    n = numel(X);
    for i = 2:n      % Knuth shuffle in forward direction:
        w = ceil(rand * i);   % 1 ≤ w ≤ i
        t = X(w);
        X(w) = X(i);
        X(i) = t;
    end
end
```

split_train_test.m

```
%%%split the train_test data
%shuffle the index, split them by test_train _ratio
%input :
%test_ratio --> portion of data for test data --> data
%output:
%test --> test_data  train -- > train_data
function [test,train] = split_train_test(data,test_ratio)
%     rng = ('default');
    num_of_observations = size(data,1);
    shuffled_indices = K_shuffle([1:num_of_observations]);
    test_set_size = floor(num_of_observations*test_ratio);
    test_indices = shuffled_indices(1:test_set_size);
    train_indices = shuffled_indices(test_set_size:end);
    test = data(test_indices,:);
    train = data(train_indices,:);

end
```

model_creation.m

```
%(critical function)
%learning the parameter for logistic regression
%input :
%data-->training_data str-->the category we want for logistic...
%regression so the data belongs to the category labeled as 1, data outside
%of the category labeled as 0(one vs all); alpha-->
%learning_rate;iter-->number of iteration for gradient decent
%output :
%theta --> trained_output J-->cost function
function [theta, J] = model_creation(data,str,alpha,iter)
    [data] = split_data_by_binary_category(data,str);
    y = str2double(data(:,1));
    x_raw = str2double(data(:,2:end));
    x = (x_raw - mean(x_raw))./std(x_raw);

    x = [ones(size(x,1),1),x];
    theta = zeros(size(x,2),1);
    m = size(x,1);
    J = 0;
    counter = 0;
    for k = 1:iter
        h = sigmoid(x*theta);
        J = (-1/m)*sum(y.*log(h)+(1-y).*log(1-h));
         counter = counter +1;
        theta = theta-(alpha/m)*x'*(h-y);
    end

end
```

split_data_by_binary_category.m

```
%marking the data_label setted by user as 1
%all other data_label are marked as zeros
%input :
%data --> data str --> data_label that user want
%marked data label
function [data] = split_data_by_binary_category(data,str)
    false_index = find(data(:,1) ≠str);
    true_index = find(data(:,1) ==str);
    data(true_index,1) = 1;
    data(false_index,1) = 0;
end
```

sigmoid.m

```
%sigmoid function
function output = sigmoid(z)
output = 1./(1+exp(-z));


end
```

predict.m

```
%%%predict the label,given the model paramters theta
%input:
%   test--> test_data;
%output:
%   result-->the output here is the numer corresponding
% to the postion of label in label array.
function [result] = predict(theta,test)
    test = str2double(test(:,2:end));
    test_data = (test - mean(test))./std(test);
    test_data = [ones(size(test,1),1),test_data];

    h = sigmoid(test_data*theta);
    [¬,col] = max(h,[],2);
    result = col;


end
```