Assignment No 2 : Introduction to machine Learning and Patter Recognition

Question number 1 :

For this problem the datasets used were all generated for a 2-dimensional real-values random vector **X** with a probability density function defined as $p(\mathbf{x})=P(L=0)P(\mathbf{x}|L=0)+P(L=1)P(\mathbf{x}|L=1)$ where L is the true class label that indicates which class-label-conditioned pdf generated the data.
The class conditional pdfs are $p(\mathbf{x}|L=0)=w_1 g(\mathbf{x}|m_{01},C_{01})+w_2 g(x|m_{02},C_{02})$ and $p(x|L=1)=g(x|m_1,C_1)$ where g(x|m,C) is a multivariate Gaussian probability density function with mean vector **m** and covariance matrix **C**. The probability functions of the classes are given below.

Class 0

$P(L=0)=0.6$

$$\mathbf{m}_{01}=\begin{bmatrix}5\\0\end{bmatrix}\quad \mathbf{C}_{01}=\begin{bmatrix}4&0\\0&2\end{bmatrix}\quad \mathbf{m}_{02}=\begin{bmatrix}0\\4\end{bmatrix}\quad \mathbf{C}_{02}=\begin{bmatrix}1&0\\0&3\end{bmatrix}$$

Class 1

$P(L=1)=0.4$

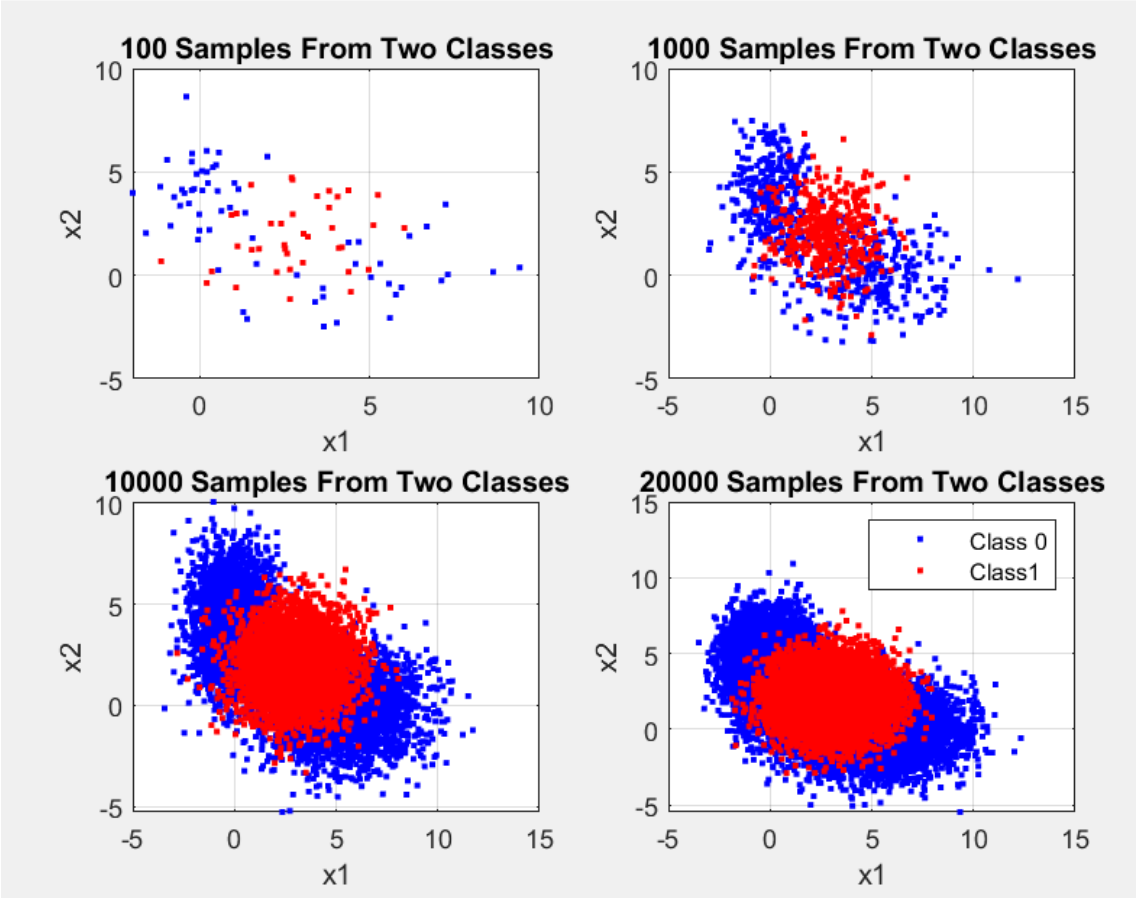
$$\mathbf{m}_1=\begin{bmatrix}3\\2\end{bmatrix}\quad \mathbf{C}_1=\begin{bmatrix}2&0\\0&2\end{bmatrix}$$

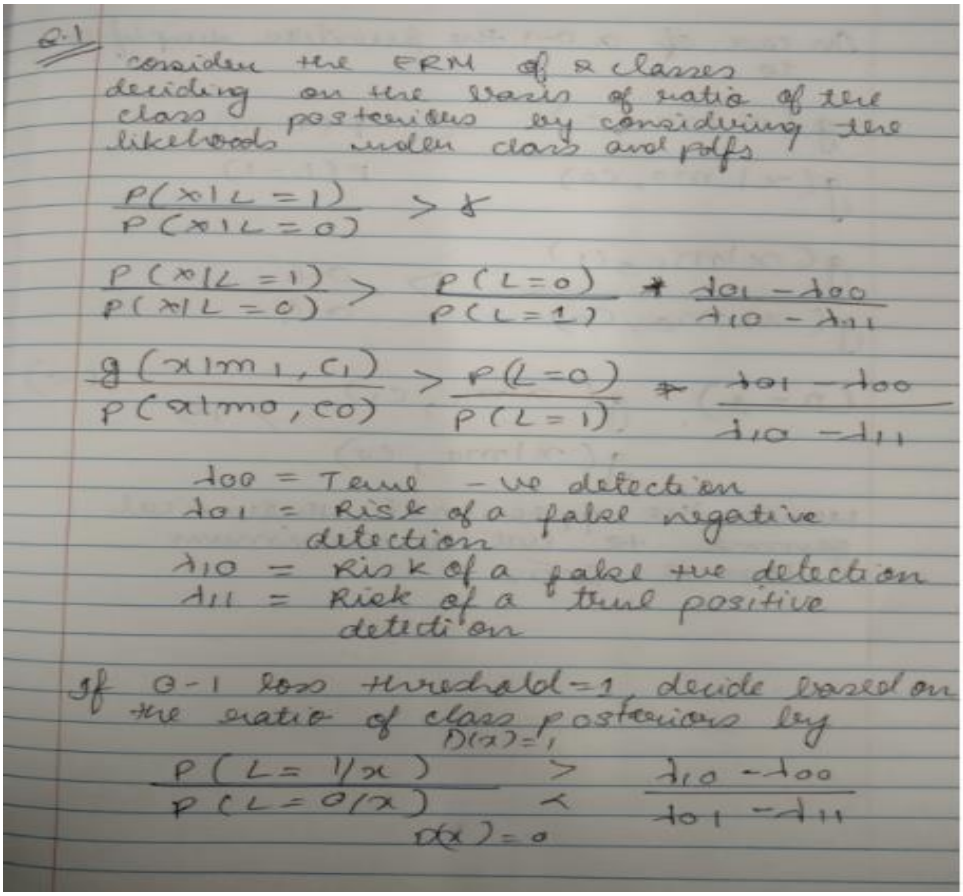
From the above distribution of X, the following datasets are generated.

- D_{train}^{100} consists of 100 samples and their labels for training;
- D_{train}^{1000} consists of 1000 samples and their labels for training;
- D_{train}^{10000} consists of 10000 samples and their labels for training;
- $D_{validate}^{20K}$ consists of 20000 samples and their labels for validation;

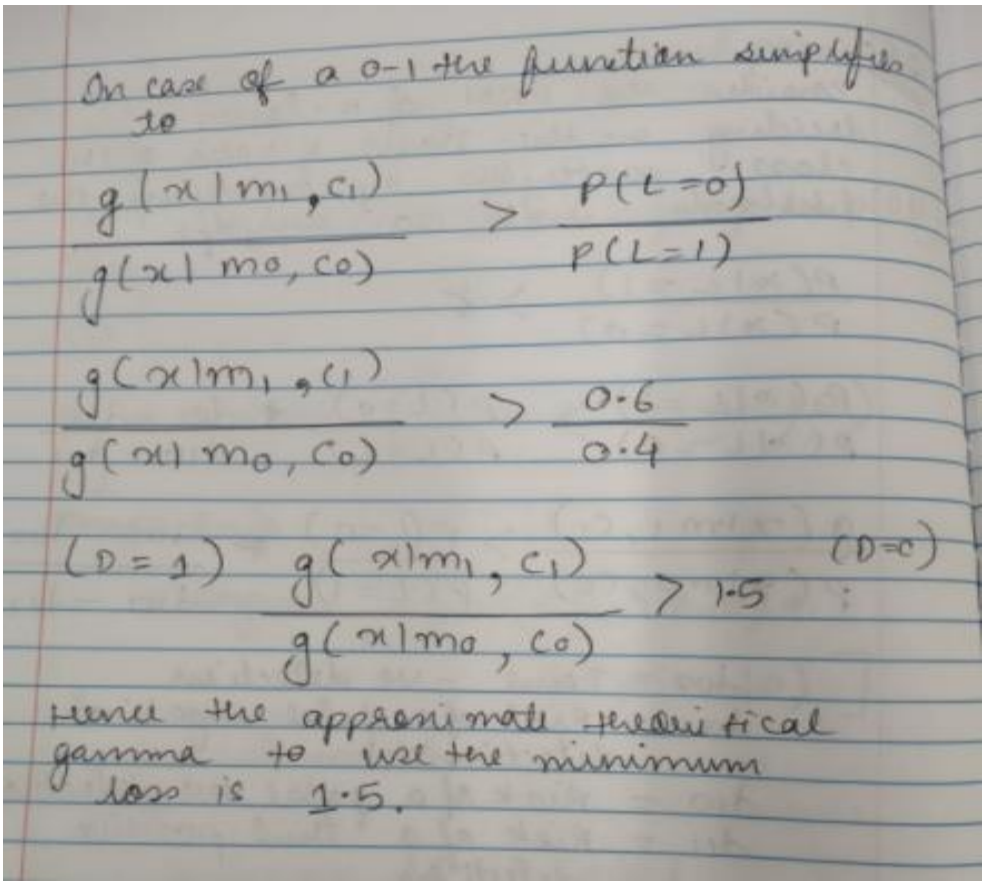
A) Determine the theoretically probability error and data set distribution as above :

The distribution of the classes is as shown below :

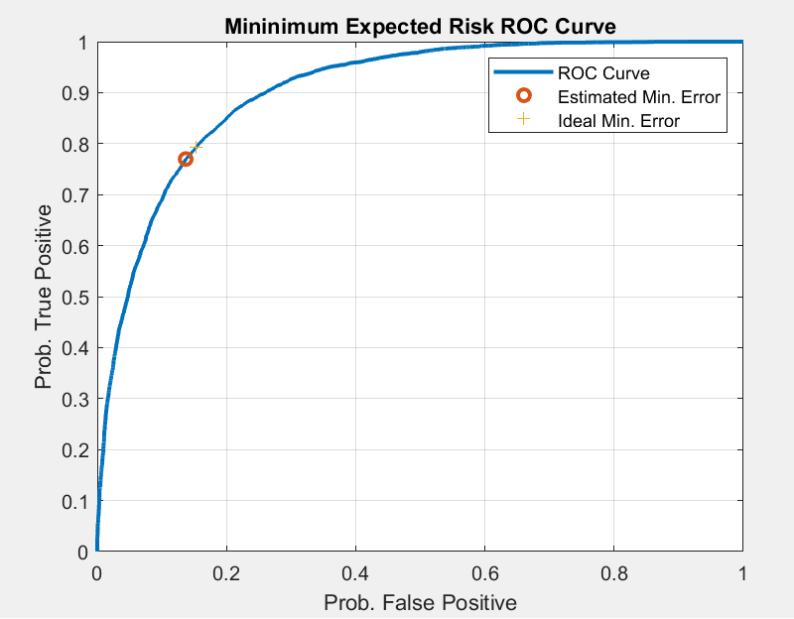

100 Samples From Two Classes
1000 Samples From Two Classes
10000 Samples From Two Classes
20000 Samples From Two Classes
x1
x2
Class 0
Class1



To minimize probability of misclassifications the cost for incorrect classification should be 1 and the cost for correct classifications should be 0 which results in the gamma shown below.



This classifier was then implemented and an estimate of ROC curve for the dataset $D^{20K}_{validate}$ was plotted. This ROC curve, along with the point of *minimum probability of error* - min(P(e)) for this dataset is shown below.



	Gamma	Min.Prob Error
Theoretical	1.5	0.799991
Estimated From Data	1.52	0.799471

B) Classifier with Estimated Parameters

For this part classification was performed with estimated knowledge of the underlying distributions of the data. Class 0 was modeled as a Gaussian Mixture Model with 2 components and Class 1 was modeled as a single Gaussian. Parameters were estimated using each of the 3 training sets of data and then the parameters estimates were used to classify the data.

Parameter estimation for Class 0 was performed using the built in Matlab function fitgmdist.

EM estimation is based on the following formulas for Maximum Likelihood Estimation of the class prior as well as the mean and covariance of the conditional pdf.

$$argmax\, Q(\theta, \theta^g) = \sum_{l=1}^{M}\sum_{i=1}^{N} \ln(\alpha_l)\, p(l|x_i\theta^g) + \sum_{l=1}^{M}\sum_{i=1}^{N} p(l|x_i\theta^g) \ln(p_l(x_i|\theta_l)$$

Maximizing that expression with respect to the individual parameters α, μ, and Σ yields

$$\hat{\alpha}_l = \frac{1}{N}\sum_{i=1}^{N} P_l(x_i|x_i, \theta^g)$$

$$\hat{u}_l = \frac{\sum_{i=1}^{N} x_i p(l|x_i\theta)}{\sum_{i=1}^{N} p(l|x_i\theta)}$$

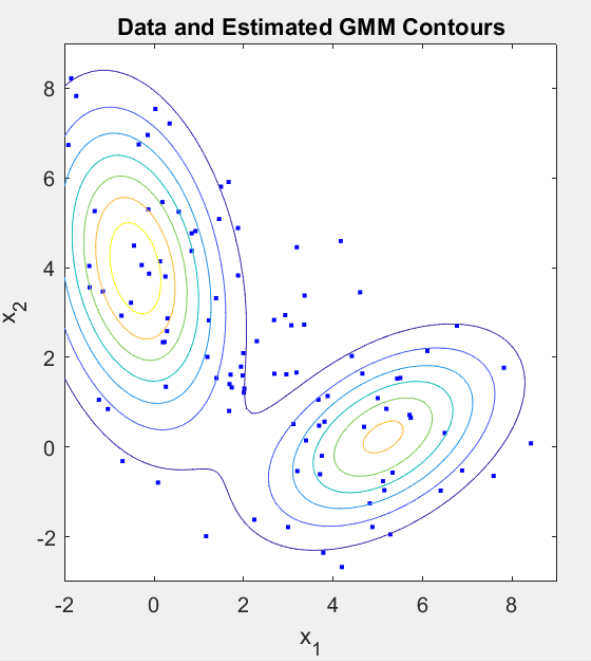
$$\hat{\Sigma}_l = \frac{\sum_{i=1}^{N} p(l|x_i\theta^g)(x_i - u_l)(x_i - u_l)^T}{\sum_{i=1}^{N} p(l|x_i\theta^g)}$$

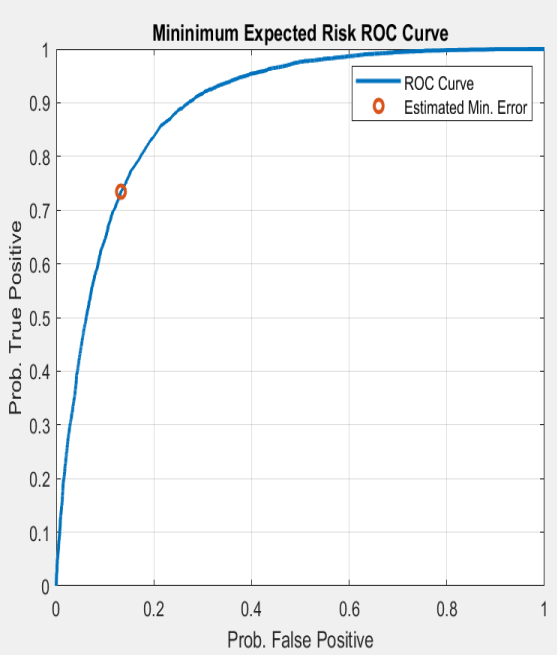
Parameter estimation for Class 1 was also performed using the fitgmdist Matlab function which makes use of the following formulas for Maximum Likelihood Estimation of the model parameters. Since there is only a single Gaussian the maximum likelihood estimates are the sample average and covariance.

$$\hat{\mu} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

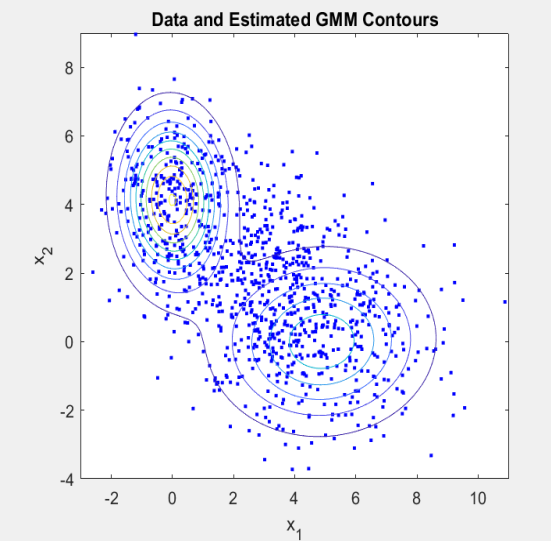
$$\hat{\Sigma} = \frac{1}{N}\sum_{i=1}^{N} (x_i - \hat{\mu})\,(x_i - \hat{\mu})^T$$

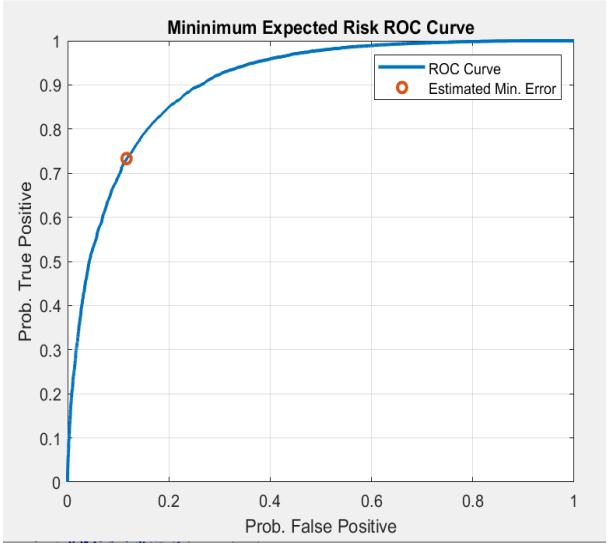
A summary of the minimum estimated probability of errors associated with the parameters estimated from the three training datasets is shown in below table . The ROC curves generated by classifying data from the validation data set based on parameters estimated from each of the training data sets.



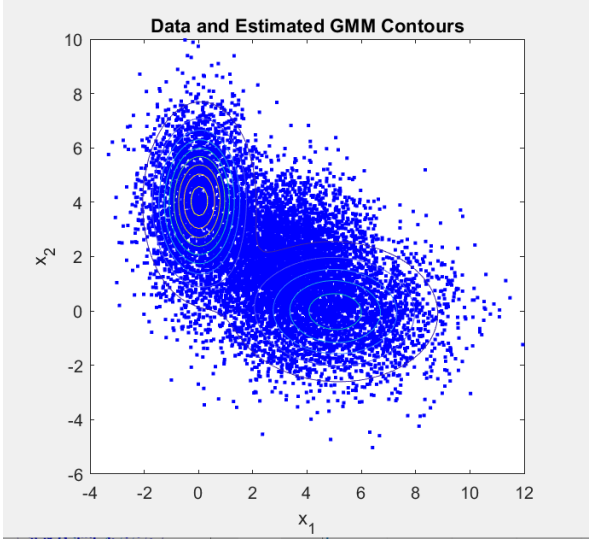


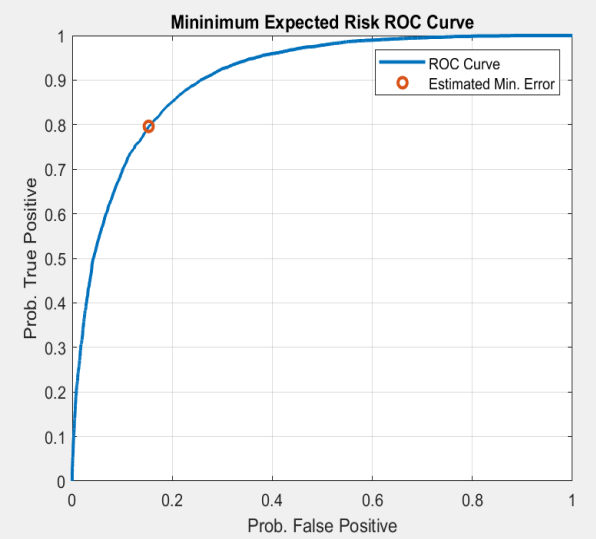
Samples 100 data roc curve





Sample 1000 roc curve and distribution





Sample 10000 roc curve and distribution

Training Samples	Min. Probability of Error	Gamma
100	**17.51%**	**2.13**
1000	**17.37%**	**1.52**
10000	**17.18%**	**1.97**

The minimum probability of error decrease as the number of training samples increases to the point where for the 10000 point training set the probability of error is comparable to the ideal case from part 1.

C) Classifier using Logistic Function

In this part maximum likelihood parameter estimation techniques were used to train logistic linear and logistic quadratic based approximation of class label posterior functions given a sample. This training was performed on each of the three separate training data sets consisting of 100, 1000, and 10000 samples respectively and was then used to classify samples from the 20000 sample validation data set.

The logistic function is defined as follows:

$$h(x,w) = \frac{1}{1+e^{w^T z(x)}}$$

For the linear logistic function $z(x)=[1\ x_1\ x_2]^T$
For the quadratic logistic function $z(x)=[1\ x_1\ x_2\ x_1^2\ x_1{*}x_2\ x_2^2]^T$
The **w** vectors are estimated using numerical optimization techniques with the cost function

$$\hat{\theta}_{ML} = -\frac{1}{N}\sum_{1}^{N} l_n \ln(h(x_n,\theta)) + (1-l_n)\ln(1-h(x_n,\theta))$$

The minimum expected risk classification criteria is then

$(l_n = 1)$ $\hat{w}^T z(x) \gtrless 0$ $(l_n = 0)$

Below table contains a summary of the resulting probability of errors from classifying the 20000 sample validation data set using each of the 3 training data sets. The data shows that for both the linear and quadratic estimation functions the probabilities of error decrease as the number of points in the training datasets increase. Additionally, the quadratic logistic function significantly outperformed the linear logistic function in all cases and for the 10000 sample training data set even approached the theoretical optimal probability of error of 0.178 obtained in Part 1.

Logistic Function Probabilities of Error

Training Dataset	Linear	Quadratic
100	44.5%	18.0%
1000	44.0%	17.8%
10000	42.1%	17.7%
20000	41.0%	17.6%

Question number 2 observations :

Given :

A vehicle at true position $[x_T, y_T]^{\mathrm{T}}$ in 2-dimensional space is to be localized using distance (range) measurements to K reference (landmark) coordinates $\{[x_1, y_1]^{\mathrm{T}}, \ldots, [x_i, y_i]^{\mathrm{T}}, \ldots, [x_K, y_K]^{\mathrm{T}}\}$. These range measurements are $r_i = d_{Ti} + n_i$ for $i \in \{1, \ldots, K\}$, where $d_{Ti} = \|[x_T, y_T]^{\mathrm{T}} - [x_i, y_i]^{\mathrm{T}}\|$ is the true distance between the vehicle and the i^{th} reference point, and n_i is a zero mean Gaussian distributed measurement noise with known variance σ_i^2. The noise in each measurement is independent from the others.

Assume that we have the following prior knowledge regarding the position of the vehicle:

$$p\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = (2\pi\sigma_x\sigma_y)^{-1} e^{-\frac{1}{2}\begin{bmatrix} x & y \end{bmatrix}\begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}^{-1}\begin{bmatrix} x \\ y \end{bmatrix}} \quad (1)$$

where $[x, y]^{\mathrm{T}}$ indicates a candidate position under consideration.

Q.2 Using formula Max likelihood parameter estimation under the additive gaussian naive Model

$$\hat{\theta}_{ML} = \text{argmax}\ p(D|\theta) = \text{argmax} \ln p(D|\theta)$$

↓ iid samples

$$= \text{argmax} \sum_{n=1}^{N} \ln p(x_n, y_n | \theta)$$

↓ Bayes Rule

$$= \text{argmax} \sum_{n=1}^{N} [\ln(p\, y_n | x_n \theta) + \ln p(x_n | \theta)]$$

↓ $x_n | \theta$ and substitute model we get

$$= \text{argmax}\ f(x|\theta) g(\theta)$$

where g is prior distribution over θ θ is random variable in Bayesian statistics. Here g is constant function.

Applying this below we get the results.

Ans Aim is to find $(x,y)^T$ coordinates position with the highest probability given the prior distribution as well as the range measurements from each of the K reference coordinates.

$$\begin{bmatrix} x_{MAP} \\ y_{MAP} \end{bmatrix} = \underset{\begin{bmatrix} x \\ y \end{bmatrix}}{\operatorname{argmax}} p\left(\begin{bmatrix} x \\ y \end{bmatrix} \mid r_1 \dots r_k\right)$$

$$= \underset{\begin{bmatrix} x \\ y \end{bmatrix}}{\operatorname{argmax}} \left((2\pi\sigma_x\sigma_y)^{-1} e^{\frac{1}{2}[x\,y]\begin{bmatrix}\sigma_x^2 & 0 \\ 0 & \sigma_y^2\end{bmatrix}^{-1}\begin{bmatrix} x \\ y \end{bmatrix}}\right) \prod_{i=1}^{K} p\left(\begin{bmatrix} x \\ y \end{bmatrix} \mid r_i\right)$$

$$\underset{[x,y]}{\operatorname{argmax}} \ln\left(2\pi\sigma_x\sigma_y\right)^{-1} + \ln e^{-\frac{1}{2}[x\,y]\begin{bmatrix}\sigma_x^2 & 0 \\ 0 & \sigma_y^2\end{bmatrix}^{-1}\begin{bmatrix} x \\ y \end{bmatrix}} + \sum_{i=1}^{k} \ln\left(p\left[\begin{matrix} x \\ y \end{matrix}\right] \mid r_i\right)$$

$$= \underset{[x,y]}{\operatorname{argmax}} -\frac{1}{2}[x\,y]\begin{bmatrix}\sigma_x^2 & 0 \\ 0 & \sigma_y^2\end{bmatrix}^{-1}\begin{bmatrix} x \\ y \end{bmatrix} + \sum_{i=1}^{k} \ln N(r_i \mid 0, \sigma_i^2)$$

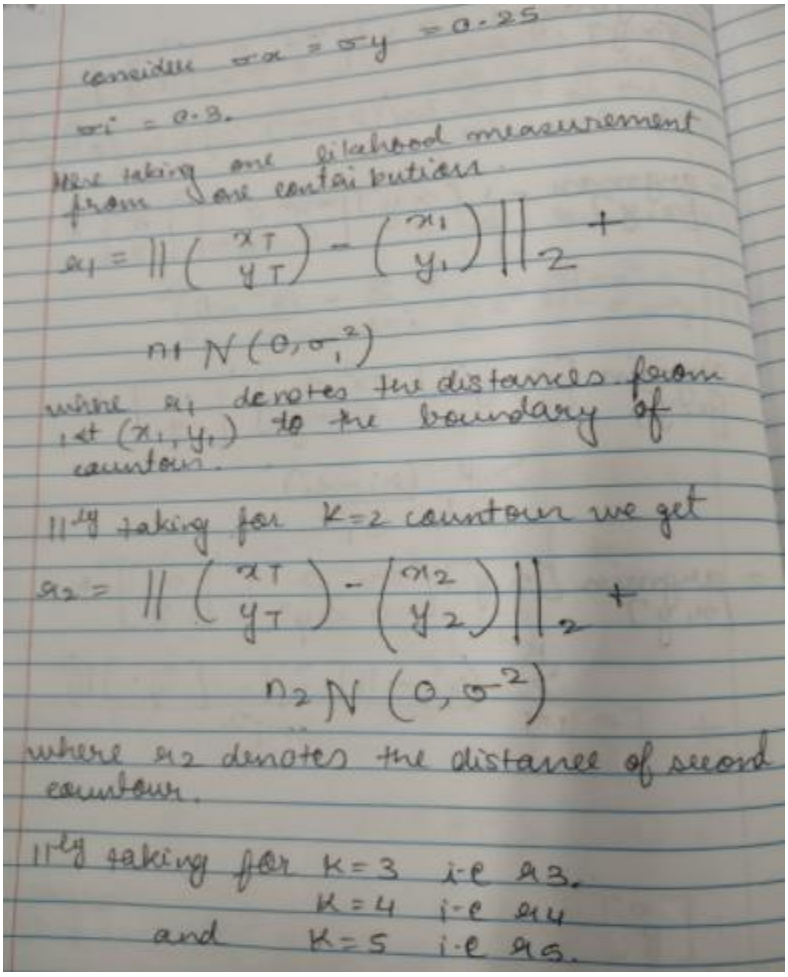
$$= \underset{[x,y]}{\operatorname{argmax}} -\frac{1}{2}[x\,y]\begin{bmatrix}\sigma_x^2 & 0 \\ 0 & \sigma_y^2\end{bmatrix}^{-1}\begin{bmatrix} x \\ y \end{bmatrix} + \sum_{i=1}^{k} \ln\left(2\pi\sigma_i^2\right)^{-\frac{1}{2}} e^{\frac{-(\|x_i - d_i\| - 0)^2}{2\sigma_i^2}}$$

$$= \underset{[x,y]}{\operatorname{argmax}} -\frac{1}{2} [x\ y] \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}^{-1} \begin{bmatrix} x \\ y \end{bmatrix} + \sum_{i=1}^{K} \ln\left((2\pi\sigma_i^2)^{-\frac{1}{2}}\right) + \ln\left(e^{\frac{-(a_i-d_i)^2}{2\sigma_i^2}}\right)$$

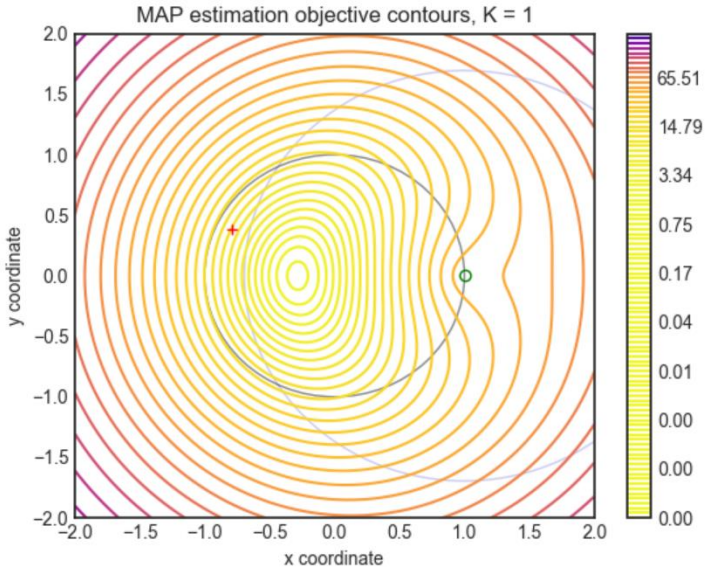
$$= \underset{[x,y]}{\operatorname{argmax}} -\frac{1}{2} [x\ y] \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}^{-1} \begin{bmatrix} x \\ y \end{bmatrix} + \sum_{i=1}^{K} -\frac{(a_i-d_i)^2}{2\sigma_i^2}$$

$$= \underset{[x,y]}{\operatorname{argmin}} [x\ y] \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}^{-1} \begin{bmatrix} x \\ y \end{bmatrix} + \sum_{i=1}^{K} \frac{(a_i-d_i)^2}{\sigma_i^2}$$

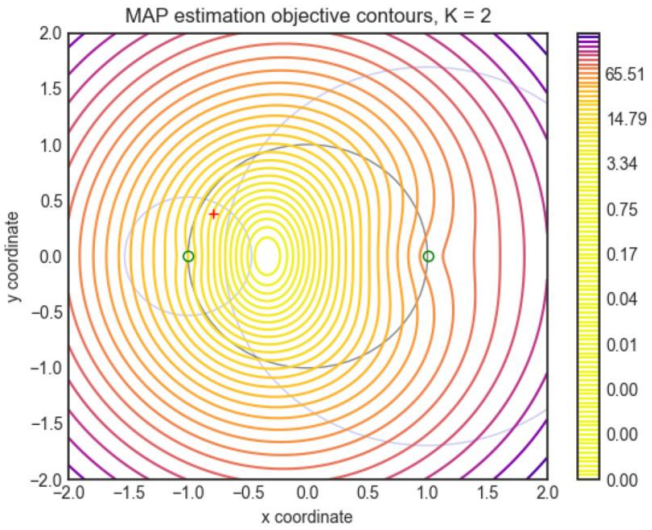
$$\underset{[x,y]}{\operatorname{argmin}} [x\ y] \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}^{-1} \begin{bmatrix} x \\ y \end{bmatrix} + \sum_{i=1}^{K} \frac{(a_i - \| \begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} x_i \\ y_i \end{bmatrix} \|)^2}{\sigma_i^2}$$


consider σx = σy = 0.25
σi = 0.3.
Here taking one likelihood measurement from one contribution
$d_1 = \left\| \begin{pmatrix} x_T \\ y_T \end{pmatrix} - \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} \right\|_2 +$
$n_1 \sim N(0, \sigma_1^2)$
where d_1 denotes the distances from i^{th} (x_1, y_1) to the boundary of countour.
11th by taking for K=2 countour we get
$d_2 = \left\| \begin{pmatrix} x_T \\ y_T \end{pmatrix} - \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} \right\|_2 +$
$n_2 \sim N(0, \sigma^2)$
where d_2 denotes the distance of second countour.
11th by taking for K = 3 i.e d_3.
K = 4 i.e d_4
and K = 5 i.e d_5.

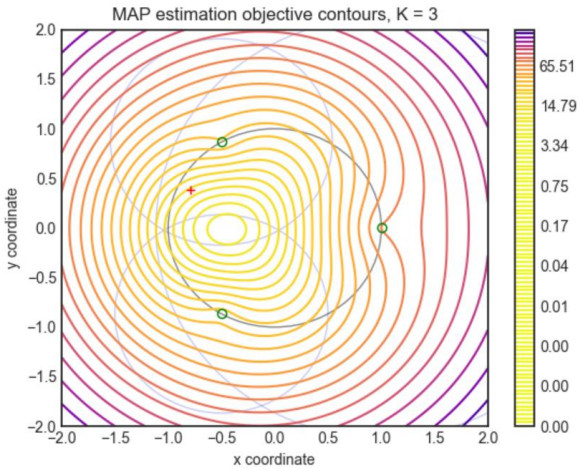
Counter K=1


MAP estimation objective contours, K = 1
y coordinate
x coordinate
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
65.51
14.79
3.34
0.75
0.17
0.04
0.01
0.00
0.00
0.00

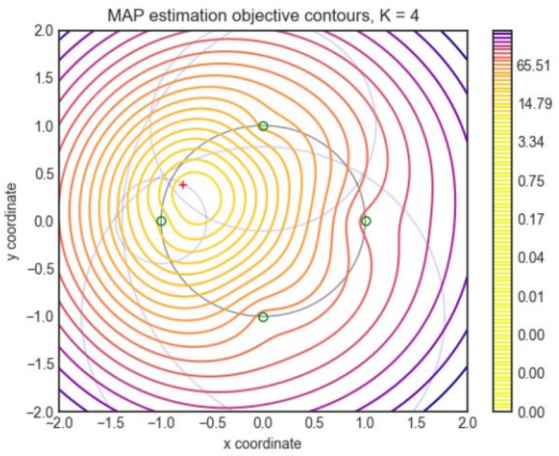
Counter K=2



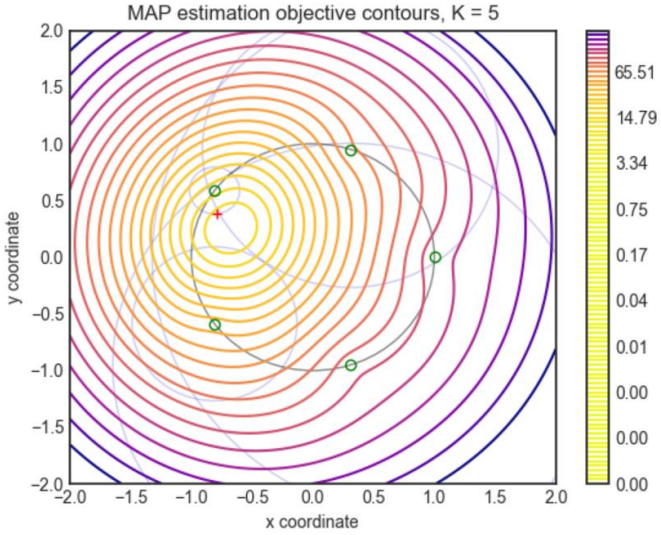
Counter 3



Counter 4



Counter 5



Explanation and observation of the above obtained counters of the vehicles estimated position:

The code gives a random coordinate pair with a unit circle at origin as true vehicle coordinates. Here from each value of K there is a defined distance from true location of the vehicles from K point that includes the gaussian noise well. Here K is evenly distributes landmarks around the circle and we are taking 5 K values.
Using MAP estimation objective function for all points 128 * 128 mesh grid equilevel contours are plotted. Gray color defines the unit circle. The true location is given by red + sign. The landmark location is given by green circles. The range of each landmark is given by blue circle for every value of the K .

The MAP estimate of position is not incredibly accurate for K < 3, where all estimates are symmetric around the x-axis because all landmarks and the prior bias have a y-coordinate of 0.

However, for K = 3 and K = 4, the estimator is much more accurate. K=5 shows most accurate estimation. This can be seen on the contour graph, in which the true location lies within only two and

one contour levels away from the central estimate contour, respectively. As we increase the size of K the MAP estimation also gets more accurate

. While this is not always true, such as in the transition from K = 1 → 2, where one of the landmark's distances measured from true landmark happened to be low enough to overcome the more accurate estimate by the new second landmark, it becomes a stronger trend as K becomes very large.

The estimator's accuracy can be determined from the contour graph based on the distance from the true location to the point with the lowest contour, roughly around the center of the innermost contour. In general, as K increases, the certainty of the estimator increases.

Landmarks whose distance is closer to the true value have more likelihood as compared to those who are far away. Likelihood is the measurement with respect to the position of the candidate standing on the unit circle.

The certainty of the estimator can be visualized on the contour graphs by a shrinkage of the area of locations with a high probability. This can be explained as every time the K increases the circle overlapping also increases. As for the figure where K=5 its shows the maximum likelihood peak because of the overlapping of 5 circles hence shrinkage is more visible and this denotes the high probability.

It is difficult to notice for such small values of K but following a single contour level as K increases (to 40 or so) demonstrates this phenomenon very well.

Question number 3 solving :

In many pattern classification problems one has the option either to assign the pattern to one of c classes, or to *reject* it as being unrecognizable. If the cost for rejects is not too high, rejection may be a desirable action. Let

$$\lambda(\alpha_i|\omega_j) = \begin{cases} 0 & i = j \qquad i, j = 1, \ldots, c \\ \lambda_r & i = c + 1 \\ \lambda_s & \text{otherwise,} \end{cases}$$

where λ_r is the loss incurred for choosing the $(c + 1)$th action, rejection, and λ_s is the loss incurred for making any substitution error. Show that the minimum risk is obtained if we decide ω_i if $P(\omega_i|\mathbf{x}) \geq P(\omega_j|\mathbf{x})$ for all j and if $P(\omega_i|\mathbf{x}) \geq 1 - \lambda_r/\lambda_s$, and reject otherwise. What happens if $\lambda_r = 0$? What happens if $\lambda_r > \lambda_s$?

Q3

Ans: A loss function states exactly how costly each action is

$$R\begin{bmatrix} D=a|x \\ \vdots \\ D=c|x \end{bmatrix} = \Lambda \begin{bmatrix} P(L=0|x) \\ P(L=1|x) \\ \vdots \\ P(L=c|x) \end{bmatrix}$$

All decisions risk across options

loss matrix function

$$\begin{bmatrix} 0 & 1s & 1s & \cdots \\ 1s & 0 & 1s & \\ 1s & 1s & 0 & \\ \vdots & & & \ddots \end{bmatrix}$$

all class posterior functions

given: $\lambda\left(\frac{\alpha_i}{w_j}\right) = \begin{cases} 0 & i=j \quad i,j=1,\ldots c \\ 1r & i=c+1 \\ 1s & \text{otherwise} \end{cases}$

Have a class $c \{ w_i, \ldots w_c \}$

Possible actions $\{\alpha_i, \ldots 2a\}$

loss function $\lambda\left(\frac{\alpha_i}{w_j}\right)$ is the loss out of which

1r is the loss incurred for choosing the $(c+1)^{th}$ action

Rejection 1s is loss incurred for making any

substitution error.

expected risk is given as

$$\text{Risk}(D(x) = d|x) = \sum_{l=1}^{c} \lambda_{dl} \underbrace{P(L=l|x)}_{\text{class posterior for label } l.}$$

where summations produces the average loss of deciding d given x considering the class posteriors probabilities for the given x

λ_{dl} denotes loss we would incur by deciding the given samples from class l.

$P(L=l|x)$ probability of class label being l given x.

so substituting loss as $\lambda\left(\frac{\omega_i}{\omega_j}\right)$ we get.

$$R(\omega_i|x) = \sum_{i=1}^{c} \lambda(\omega_i|\omega_j) P(\omega_j|x)$$

$$i = 1, 2, \ldots c \rightarrow (1)$$

$$R(\omega_{c+1}|x) = \lambda_{re} \rightarrow (2)$$

Now for $i = 1, 2, \ldots c$ the eq is can be summarised as below:

$$R(\alpha_i|x) = \lambda_s \sum_{j=1, j\neq i}^{c} \lambda(\alpha_i|w_j) P(w_j|x)$$

$$R(\alpha_i|x) = \lambda_s (1 - P(w_i|x)) \rightarrow (3)$$

our decision function is to choose the action with the function with lowest risk function, which implies we only reject the options if and only if

$$\lambda_r > \lambda_s (1 - P(w_i|x)) \quad \forall \; i = 1, 2, 3, \ldots$$

this is same as

$$P(w_i|x) < 1 - \frac{\lambda_r}{\lambda_s} \quad \forall \; i = 1, 2, 3, \ldots c$$

This means that class with the highest posterior probabilities is choosen as a threshold $\left(1 - \frac{\lambda_r}{\lambda_s}\right)$ we must reject

To show minimum risk probability

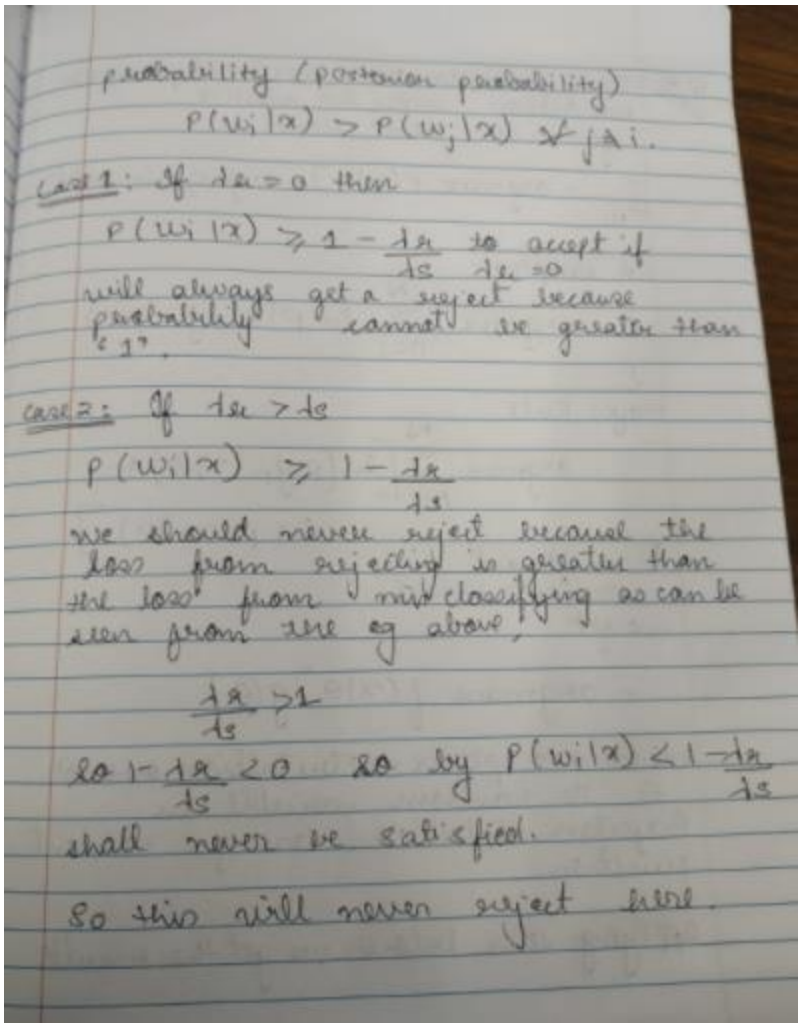
$$R(\alpha_i|x) \leq R(\alpha_{c+1}|x)$$

$$\lambda_s (1 - P(w_i|x)) \leq \lambda_r$$

$$1 - P(w_i|x) \leq \frac{\lambda_r}{\lambda_s}$$

$$1 - \frac{\lambda_r}{\lambda_s} \leq P(w_i|x)$$

provide the class is choosen with highest



Question 1 code : MATLAB

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Significant parts of this code were derived from the following sources
%g/practice/EMforGMM.m
%g/practice/evalGaussian.m
%g/practice/generateDataFromGMM.m
%g/practice/randGaussian.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all;
close all;
%Switches to bypass parts 1 and 2 for debugging
Part1=1;Part2=1;
dimension=2; %Dimension of data
%Define data
D.d100.N=100;
D.d1000.N=1000;
D.d10k.N=10000;
```

```
D.d20k.N=20000;
dTypes=fieldnames(D);
%Define Statistics
p=[0.6 0.4]; %Prior
%Label 0 GMM Stats
mu0=[5 0;0 4]';
Sigma0(:,:,1)=[4 0;0 2];
Sigma0(:,:,2)=[1 0;0 3];
alpha0=[0.5 0.5];
%Label 1 Single Gaussian Stats
mu1=[3 2]';
Sigma1=[2 0;0 2];
alpha1=1;
figure;
%Generate Data
for ind=1:length(dTypes)
 D.(dTypes{ind}).x=zeros(dimension,D.(dTypes{ind}).N); %Initialize Data

 %Determine Posteriors
 D.(dTypes{ind}).labels = rand(1,D.(dTypes{ind}).N)>=p(1);
 D.(dTypes{ind}).N0=sum(~D.(dTypes{ind}).labels);
 D.(dTypes{ind}).N1=sum(D.(dTypes{ind}).labels);
 D.(dTypes{ind}).phat(1)=D.(dTypes{ind}).N0/D.(dTypes{ind}).N;
 D.(dTypes{ind}).phat(2)=D.(dTypes{ind}).N1/D.(dTypes{ind}).N;
 [D.(dTypes{ind}).x(:,~D.(dTypes{ind}).labels),...
 D.(dTypes{ind}).dist(:,~D.(dTypes{ind}).labels)]=...
 randGMM(D.(dTypes{ind}).N0,alpha0,mu0,Sigma0);
 [D.(dTypes{ind}).x(:,D.(dTypes{ind}).labels),...
 D.(dTypes{ind}).dist(:,D.(dTypes{ind}).labels)]=...
 randGMM(D.(dTypes{ind}).N1,alpha1,mu1,Sigma1);
 subplot(2,2,ind);
 plot(D.(dTypes{ind}).x(1,~D.(dTypes{ind}).labels),...
 D.(dTypes{ind}).x(2,~D.(dTypes{ind}).labels),'b.','DisplayName','Class 0');
 hold all;
 plot(D.(dTypes{ind}).x(1,D.(dTypes{ind}).labels),...
 D.(dTypes{ind}).x(2,D.(dTypes{ind}).labels),'r.','DisplayName','Class1');
 grid on;
 xlabel('x1');ylabel('x2');
 title([num2str(D.(dTypes{ind}).N) ' Samples From Two Classes']);

end
legend 'show';
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Part 1: Optimal Classifier with Knowledge of PDFs
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if Part1

px0=evalGMM(D.d20k.x,alpha0,mu0,Sigma0);
px1=evalGaussian(D.d20k.x ,mu1,Sigma1);
discScore=log(px1./px0);
sortDS=sort(discScore);
%Generate vector of gammas for parametric sweep
```

```
logGamma=[min(discScore)-eps sort(discScore)+eps];
prob=CalcProb(discScore,logGamma,D.d20k.labels,D.d20k.N0,D.d20k.N1,D.d20k.phat);
logGamma_ideal=log(p(1)/p(2));
decision_ideal=discScore>logGamma_ideal;
p10_ideal=sum(decision_ideal==1 & D.d20k.labels==0)/D.d20k.N0;
p11_ideal=sum(decision_ideal==1 & D.d20k.labels==1)/D.d20k.N1;
pFE_ideal=(p10_ideal*D.d20k.N0+(1-p11_ideal)*D.d20k.N1)/(D.d20k.N0+D.d20k.N1);
%Estimate Minimum Error
%If multiple minimums are found choose the one closest to the theoretical
%minimum
[prob.min_pFE, prob.min_pFE_ind]=min(prob.pFE);
if length(prob.min_pFE_ind)>1
[~,minDistTheory_ind]=min(abs(logGamma(prob.min_pFE_ind)-logGamma_ideal));
prob.min_pFE_ind=prob.min_pFE_ind(minDistTheory_ind);
end
%Find minimum gamma and corresponding false and true positive rates
minGAMMA=exp(logGamma(prob.min_pFE_ind));
prob.min_FP=prob.p10(prob.min_pFE_ind);
prob.min_TP=prob.p11(prob.min_pFE_ind);
%Plot
plotROC(prob.p10,prob.p11,prob.min_FP,prob.min_TP);
hold all;
plot(p10_ideal,p11_ideal,'+','DisplayName','Ideal Min. Error');
plotMinPFE(logGamma,prob.pFE,prob.min_pFE_ind);
plotDecisions(D.d20k.x,D.d20k.labels,decision_ideal);
fprintf('m = %f\n ',prob.p10);
fprintf('m = %f\n', prob.p11);
fprintf('m = %f\n', prob.min_FP);
fprintf('m = %f\n', prob.min_TP);
%plotERMContours(D.d10k.x,alpha0,mu0,Sigma0,mu1,Sigma1,logGamma_ideal);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Part 2: Classificiation with Maximumlikelihood Parameter Estimation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if Part2
for ind=1:length(dTypes)-1

 %Estimate Parameters using matlab built in function
 D.(dTypes{ind}).DMM_Est0=...
fitgmdist(D.(dTypes{ind}).x(:,~D.(dTypes{ind}).labels)',2,'Replicates',10);
 D.(dTypes{ind}).DMM_Est1=...
 fitgmdist(D.(dTypes{ind}).x(:,D.(dTypes{ind}).labels)',1);

 plotContours(D.(dTypes{ind}).x,...
 D.(dTypes{ind}).DMM_Est0.ComponentProportion,...
 D.(dTypes{ind}).DMM_Est0.mu,D.(dTypes{ind}).DMM_Est0.Sigma);

 %Calculate discriminate score
 px0=pdf(D.(dTypes{ind}).DMM_Est0,D.d10k.x');
 px1=pdf(D.(dTypes{ind}).DMM_Est1,D.d10k.x');
 discScore=log(px1'./px0');
```

```
 sortDS=sort(discScore);
 %Generate vector of gammas for parametric sweep
 logGamma=[min(discScore)-eps sort(discScore)+eps];
 prob=CalcProb(discScore,logGamma,D.d20k.labels,...
 D.d20k.N0,D.d20k.N1,D.(dTypes{ind}).phat);

 %Estimate Minimum Error
 %If multiple minimums are found choose the one closest to the theoretical
 %minimum
 [prob.min_pFE, prob.min_pFE_ind]=min(prob.pFE);
 if length(prob.min_pFE_ind)>1
 [~,minDistTheory_ind]=...
 min(abs(logGamma(prob.min_pFE_ind)-logGamma_ideal));
 prob.min_pFE_ind=min_pFE_ind(minDistTheory_ind);
 end

 %Find minimum gamma and corresponding false and true positive rates
 minGAMMA=exp(logGamma(prob.min_pFE_ind));
 prob.min_FP=prob.p10(prob.min_pFE_ind);
 prob.min_TP=prob.p11(prob.min_pFE_ind);

 %Plot
 plotROC(prob.p10,prob.p11,prob.min_FP,prob.min_TP);
 plotMinPFE(logGamma,prob.pFE,prob.min_pFE_ind);
 fprintf('Estimated: Gamma=%1.2f, Error=%1.2f%%\n',...
 minGAMMA,100*prob.min_pFE);


end

end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Part 3: Classificiation with Maximumlikelihood Parameter Estimation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
options=optimset('MaxFunEvals',3000,'MaxIter',1000);
for ind=1:length(dTypes)
 lin.x=[ones(1,D.(dTypes{ind}).N); D.(dTypes{ind}).x];
 lin.init=zeros(dimension+1,1);
 %[lin.theta,lin.cost]=thetaEst(lin.x,lin.init,D.(dTypes{ind}).labels);
[lin.theta,lin.cost]=fminsearch(@(theta)(costFun(theta,lin.x,D.(dTypes{ind}).labe
ls)), lin.init,options);
lin.discScore=lin.theta'*[ones(1,D.d20k.N); D.d10k.x];
 gamma=0;

 lin.prob=CalcProb(lin.discScore,gamma,D.d20k.labels,...
 D.d20k.N0,D.d20k.N1,D.d20k.phat);

 % quad.decision=[ones(D.d10k.N,1) D.d10k.x]*quad.theta>0;

 plotDecisions(D.d20k.x,D.d20k.labels,lin.prob.decisions);
 title(sprintf('Data and Classifier Decisions Against True Label for Linear
Logistic Fit\nProbability of Error=%1.1f%%',100*lin.prob.pFE));
```

```
 % plotDecisions(D.d10k.x,D.d10k.labels,quad.decision);

 quad.x=[ones(1,D.(dTypes{ind}).N); D.(dTypes{ind}).x;...
 D.(dTypes{ind}).x(1,:).^2;...
 D.(dTypes{ind}).x(1,:).*D.(dTypes{ind}).x(2,:);...
 D.(dTypes{ind}).x(2,:).^2];
 quad.init= zeros(2*(dimension+1),1);

 [quad.theta,quad.cost]=...
 fminsearch(@(theta)(costFun(theta,quad.x,D.(dTypes{ind}).labels)),...
 quad.init,options);

 quad.xScore=[ones(1,D.d20k.N); D.d20k.x; D.d20k.x(1,:).^2;...
 D.d20k.x(1,:).*D.d20k.x(2,:); D.d20k.x(2,:).^2];

 quad.discScore=quad.theta'*quad.xScore;
 gamma=0;
 quad.prob=CalcProb(quad.discScore,gamma,D.d20k.labels,...
 D.d20k.N0,D.d20k.N1,D.d20k.phat);

 plotDecisions(D.d20k.x,D.d20k.labels,quad.prob.decisions);
 title(sprintf('Data and Classifier Decisions Against True Label for Quadratic
Logistic Fit\nProbability of Error=%1.1f%%',100*quad.prob.pFE));

end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Function Definitions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function cost=costFun(theta,x,labels)
h=1./(1+exp(-x'*theta));

cost=-1/length(h)*sum((labels'.*log(h)+(1-labels)'.*(log(1-h))));
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [x,labels] = randGMM(N,alpha,mu,Sigma)
d = size(mu,1); % dimensionality of samples
cum_alpha = [0,cumsum(alpha)];
u = rand(1,N); x = zeros(d,N); labels = zeros(1,N);
for m = 1:length(alpha)
 ind = find(cum_alpha(m)<u & u<=cum_alpha(m+1));
 x(:,ind) = randGaussian(length(ind),mu(:,m),Sigma(:,:,m));
 labels(ind)=m-1;
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function x = randGaussian(N,mu,Sigma)
% Generates N samples from a Gaussian pdf with mean mu covariance Sigma
n = length(mu);
z = randn(n,N);
A = Sigma^(1/2);
x = A*z + repmat(mu,1,N);
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function gmm = evalGMM(x,alpha,mu,Sigma)
gmm = zeros(1,size(x,2));
for m = 1:length(alpha) % evaluate the GMM on the grid
 gmm = gmm + alpha(m)*evalGaussian(x,mu(:,m),Sigma(:,:,m));
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function g = evalGaussian(x,mu,Sigma)
% Evaluates the Gaussian pdf N(mu,Sigma) at each coumn of X
[n,N] = size(x);
invSigma = inv(Sigma);
C = (2*pi)^(-n/2) * det(invSigma)^(1/2);
E = -0.5*sum((x-repmat(mu,1,N)).*(invSigma*(x-repmat(mu,1,N))),1);
g = C*exp(E);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function prob=CalcProb(discScore,logGamma,labels,N0,N1,phat)
for ind=1:length(logGamma)
 prob.decisions=discScore>=logGamma(ind);
 Num_pos(ind)=sum(prob.decisions);
 prob.p10(ind)=sum(prob.decisions==1 & labels==0)/N0;
 prob.p11(ind)=sum(prob.decisions==1 & labels==1)/N1;
 prob.p01(ind)=sum(prob.decisions==0 & labels==1)/N1;
 prob.p00(ind)=sum(prob.decisions==0 & labels==0)/N0;

 prob.pFE(ind)=prob.p10(ind)*phat(1) + prob.p01(ind)*phat(2);
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function plotContours(x,alpha,mu,Sigma)
figure
if size(x,1)==2
 plot(x(1,:),x(2,:),'b.');
 xlabel('x_1'), ylabel('x_2'), title('Data and Estimated GMM Contours'),
 axis equal, hold on;
 rangex1 = [min(x(1,:)),max(x(1,:))];
 rangex2 = [min(x(2,:)),max(x(2,:))];
 [x1Grid,x2Grid,zGMM] = contourGMM(alpha,mu,Sigma,rangex1,rangex2);
 contour(x1Grid,x2Grid,zGMM); axis equal,
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [x1Grid,x2Grid,zGMM] = contourGMM(alpha,mu,Sigma,rangex1,rangex2)
x1Grid = linspace(floor(rangex1(1)),ceil(rangex1(2)),101);
x2Grid = linspace(floor(rangex2(1)),ceil(rangex2(2)),91);
[h,v] = meshgrid(x1Grid,x2Grid);
GMM = evalGMM([h(:)';v(:)'],alpha, mu, Sigma);
zGMM = reshape(GMM,91,101);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function plotROC(p10,p11,min_FP,min_TP)
figure;
plot(p10,p11,'DisplayName','ROC Curve','LineWidth',2);
hold all;
plot(min_FP,min_TP,'o','DisplayName','Estimated Min. Error','LineWidth',2);
xlabel('Prob. False Positive');
ylabel('Prob. True Positive');
title('Mininimum Expected Risk ROC Curve');
legend 'show';
grid on; box on;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function plotMinPFE(logGamma,pFE,min_pFE_ind)
figure;
plot(logGamma,pFE,'DisplayName','Errors','LineWidth',2);
hold on;
plot(logGamma(min_pFE_ind),pFE(min_pFE_ind),...
 'ro','DisplayName','Minimum Error','LineWidth',2);
xlabel('Gamma');
ylabel('Proportion of Errors');
title('Probability of Error vs. Gamma')
grid on;
legend 'show';
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function plotDecisions(x,labels,decisions)
ind00 = find(decisions==0 & labels==0);
ind10 = find(decisions==1 & labels==0);
ind01 = find(decisions==0 & labels==1);
ind11 = find(decisions==1 & labels==1);
figure; % class 0 circle, class 1 +, correct green, incorrect red
end
```

Question 2 code :

```
#!/usr/bin/env python3

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import LogNorm

SIGMA_X = 0.25
SIGMA_Y = 0.25
SIGMA_I = 0.3

CONTOUR_LEVELS = np.geomspace(0.0001, 250, 100)

# Returns a random xy pairs within the unit circle
```

```
def randomcirclevalues():
    # Polar coordinates with a square-rooted r-value that generates uniform
distribution
    r = np.sqrt(np.random.uniform(0, 1)) # radius is one as its a unit circle
    theta = np.random.uniform(0, 2) * np.pi
    x = r * np.cos(theta)        #x coordinates
    y = r * np.sin(theta)        #y coordinates
    return np.array([x, y])

# Returns the xy coordinate pair of the i'th landmark out of K total landmarks.
def counterK_pos(i, K):
    angle = 2 * np.pi / K * i
    x = np.cos(angle)
    y = np.sin(angle)
    return np.array([x, y])


# Generates a distance ri between the given landmark and true positions,
# with random Gaussian noise included
def distancewithnoise(xy_landmark, xy_true):
    dTi = np.linalg.norm(xy_true-xy_landmark)
    while True:
        noise = np.random.normal(0, SIGMA_I)
        distance = dTi + noise
        if distance >= 0:
            return distance


# Generates noisy gaussian for all K landmarks given the true position.
def totalmeasurements(K, xy_true):
    return [distancewithnoise(counterK_pos(i, K), xy_true) for i in range(K)]

# Creates an equilevel contour plot for the MAP estimation objective function
# given a set of distance measurements
def equallevel(distance_measurements, xy_true):
    #  creating a mesh grid of values from the objective function
    gridpoints = np.meshgrid(np.linspace(-2, 2, 128), np.linspace(-2, 2, 128))
    contour_values = MAP_estimation(gridpoints, distance_measurements)

    # Setting plot
    plt.style.use('seaborn-white')

    ax = plt.gca()

    unit_circle = plt.Circle((0, 0), 1, color='#888888', fill=False)
```

```
    ax.add_artist(unit_circle)

    plt.contour(gridpoints[0], gridpoints[1], contour_values, cmap='plasma_r',
levels=CONTOUR_LEVELS);

    for (i, r_i) in enumerate(distance_measurements):
        (x, y) = counterK_pos(i, len(distance_measurements))
        plt.plot((x), (y), 'o', color='g', markerfacecolor='none')
        # blue colour defines the distance from each landmark
        range_circle = plt.Circle((x, y), r_i, color='#0000bb33', fill=False)
        ax.add_artist(range_circle)

    ax.set_xlabel("x coordinate")
    ax.set_ylabel("y coordinate")
    ax.set_title("MAP estimation objective contours, K = " +
str(len(distance_measurements)))

    ax.set_xlim((-2, 2))
    ax.set_ylim((-2, 2))
    ax.plot([xy_true[0]], [xy_true[1]], '+', color='r')
    plt.colorbar();
    plt.show()

# Calculates values of the MAP estimation objective function on a given mesh
# grid of input x-y coordinate pairs.
def MAP_estimation(xy, distance_measurements):
    # The shape of xy is (2, n, m), but it needs to be (n, m, 1, 2).
    xy = np.expand_dims(np.transpose(xy, axes=(1, 2, 0)), axis=len(np.shape(xy))-
1)

    prior = np.matmul(xy, np.linalg.inv(np.array([[SIGMA_X**2, 0],[0,
SIGMA_Y**2]])))
    prior = np.matmul(prior, np.swapaxes(xy, 2, 3))
    prior = np.squeeze(prior)
    # prior is now of shape (n, m).

    range_sum = 0

    for (i, r_i) in enumerate(distance_measurements):
        xy_i = counterK_pos(i, len(distance_measurements))
        d_i = np.linalg.norm(xy - xy_i[None, None, None, :], axis=3)
        range_sum += np.squeeze((r_i - d_i)**2 / SIGMA_I**2)

    return prior + range_sum    # give the maximum likehood parameter
```

```
xy_true = randomcirclevalues()

for K in [1, 2, 3, 4, 5]:
    distance_measurements = totalmeasurements(K, xy_true)
    equallevel(distance_measurements, xy_true)
```