EECE5644 HW2

CONTENT

Question 1

The probability functions of the classes are given below.

$$\text{Class } 0: P(L=0) = 0.6, m_{01} = \begin{bmatrix}5\\0\end{bmatrix} \; c_{01} = \begin{bmatrix}4 & 0\\0 & 2\end{bmatrix} \; w_1 = 0.5$$

$$\text{And} \quad m_{02} = \begin{bmatrix}0\\4\end{bmatrix} \; c_{02} = \begin{bmatrix}1 & 0\\0 & 3\end{bmatrix} \; w_2 = 0.5$$

$$\text{Class } 1: P(L=1) = 0.4, m_1 = \begin{bmatrix}3\\2\end{bmatrix} \; c_1 = \begin{bmatrix}2 & 0\\0 & 2\end{bmatrix}$$

For the following exercises 4 sets of data were generated:

- D_{train}^{100} consisting of 100 samples and their labels for training
- D_{train}^{1000} consisting of 1000 samples and their labels for training
- D_{train}^{10000} consisting of 10000 samples and their labels for training
- $D_{Validate}^{20k}$ consisting of 20000 samples and their labels for validation

Plots for the datasets used are shown below in Figure 1.

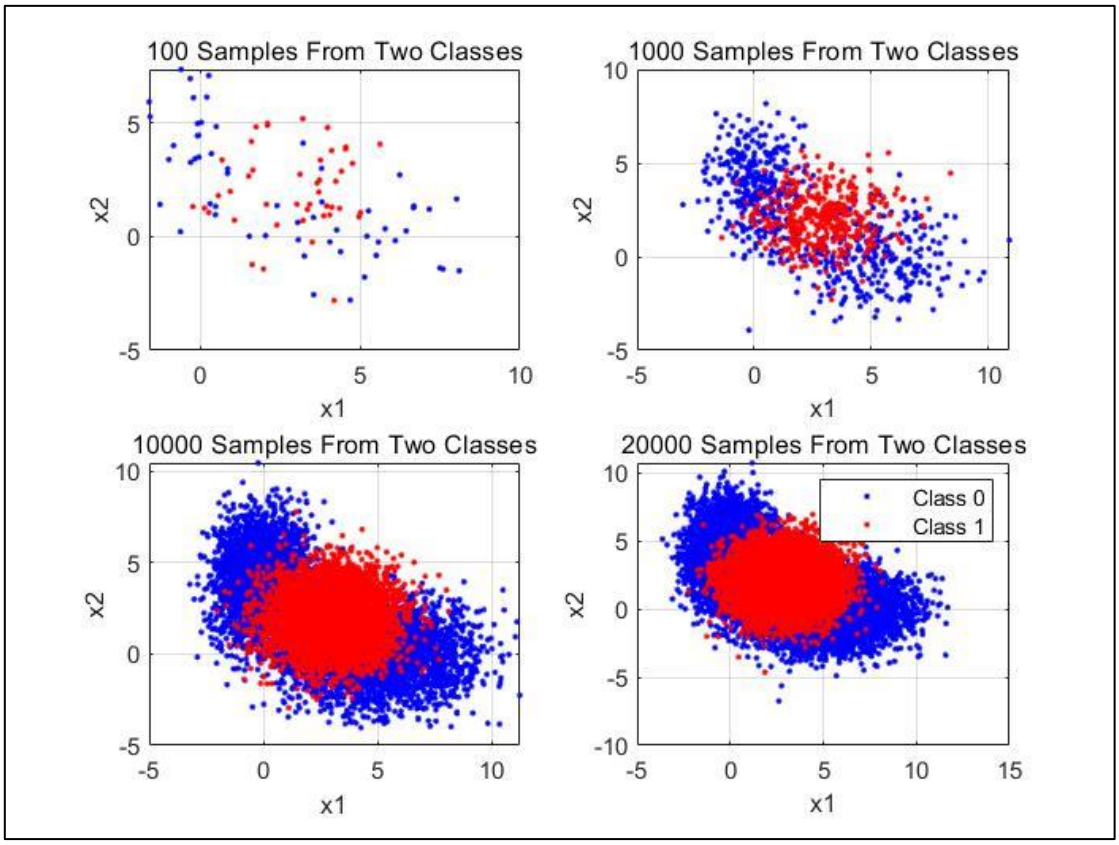

Figure 1: Question 1 Training and Validation Data

Part 1

The minimum expected risk classification rule:

$$g(x \mid m_0, c_0) = w_1 g(x \mid m_{01}, c_{01}) + w_2 g(x \mid m_{02}, c_{02})$$

$$(D = 1) \quad \frac{g(x \mid m_0, c_0)}{g(x \mid m_1, c_1)} \gtrless \frac{P(L = 0)}{P(L = 1)} \left(\frac{\lambda_{10} - \lambda_{00}}{\lambda_{01} - \lambda_{11}}\right) = \frac{0.6}{0.4} \left(\frac{\lambda_{10} - \lambda_{00}}{\lambda_{01} - \lambda_{11}}\right) \quad (D = 0)$$

$$\text{That is} \quad \frac{w_1 g(x|m_{01}, c_{01}) + w_2 g(x|m_{02}, c_{02})}{g(x|m_1, c_1)} \gtrless 1.5 \left(\frac{\lambda_{10} - \lambda_{00}}{\lambda_{01} - \lambda_{11}}\right)$$

The classifier was implemented for multiple values of gamma and the ROC curve is shown in Figure 2 below. The locations of the theoretical minimum error (orange plus) as well as the minimum error determined by a parametric sweep of gamma (red circle) are marked on the plot.

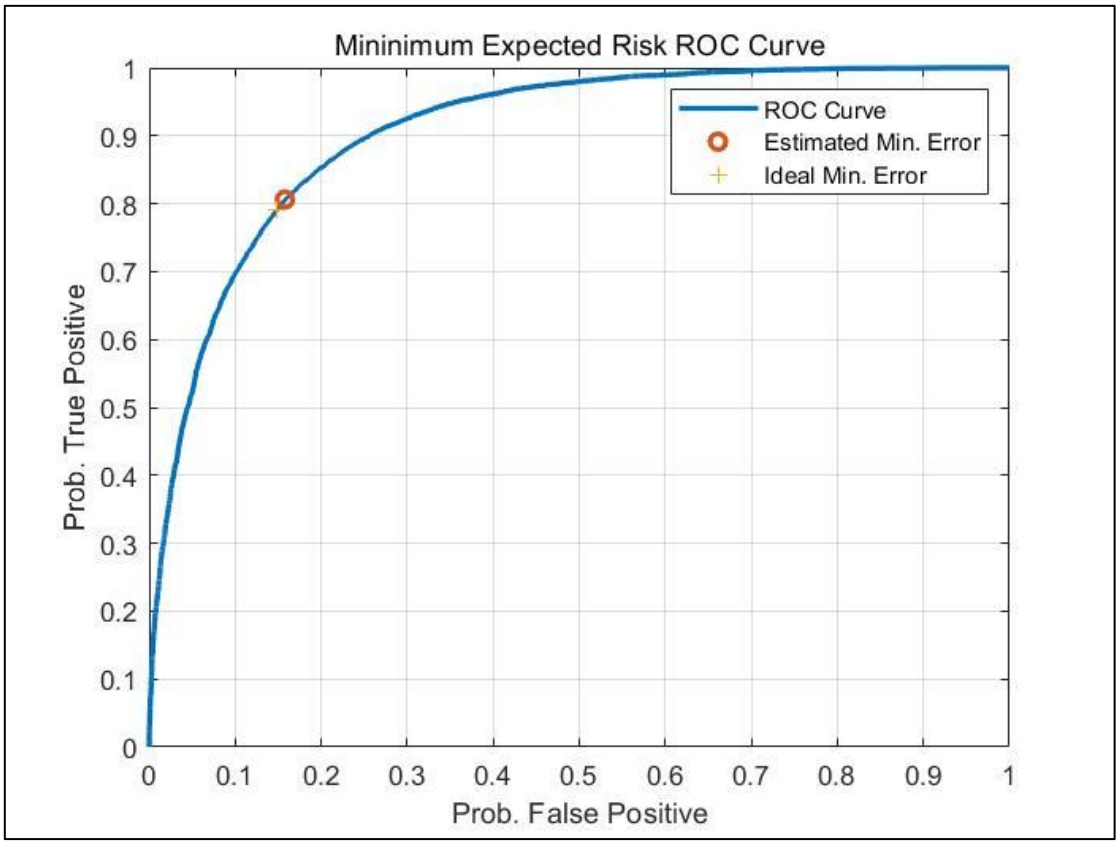

Figure 2: ROC Curve for Known Ideal Classification Case

Table 1 compares the theoretical and the calculated minimum probability of error. As can be seen the two values closely align providing confidence in the estimated value. 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.

$$(D=1)\quad \frac{P(x \mid \mathrm{L0})}{p(x \mid \mathrm{L1})} \gtrless \frac{1-0}{1-0} * \frac{0.6}{0.4} = 1.5 = \gamma \quad (D=0)$$

Plots of the ROC curve with the calculated ideal minimum error point as well as the minimum error point estimated from the generated validation data is shown in Figure 2. The probability of error versus Gamma with the calculated and estimated minimum error points marked are shown in Figure 3. The minimum errors associated with the calculated and estimated cases along with their associated gammas are shown in Table 1. As can be seen in the plots and table both the calculated and estimated minimum error points are in near agreement.

Table 1: Comparison of Gammas to Produce Minimum Errors

	γ	min P_e
Theoretical	1.50	0.1746
Calculated from data	1.53	0.1729

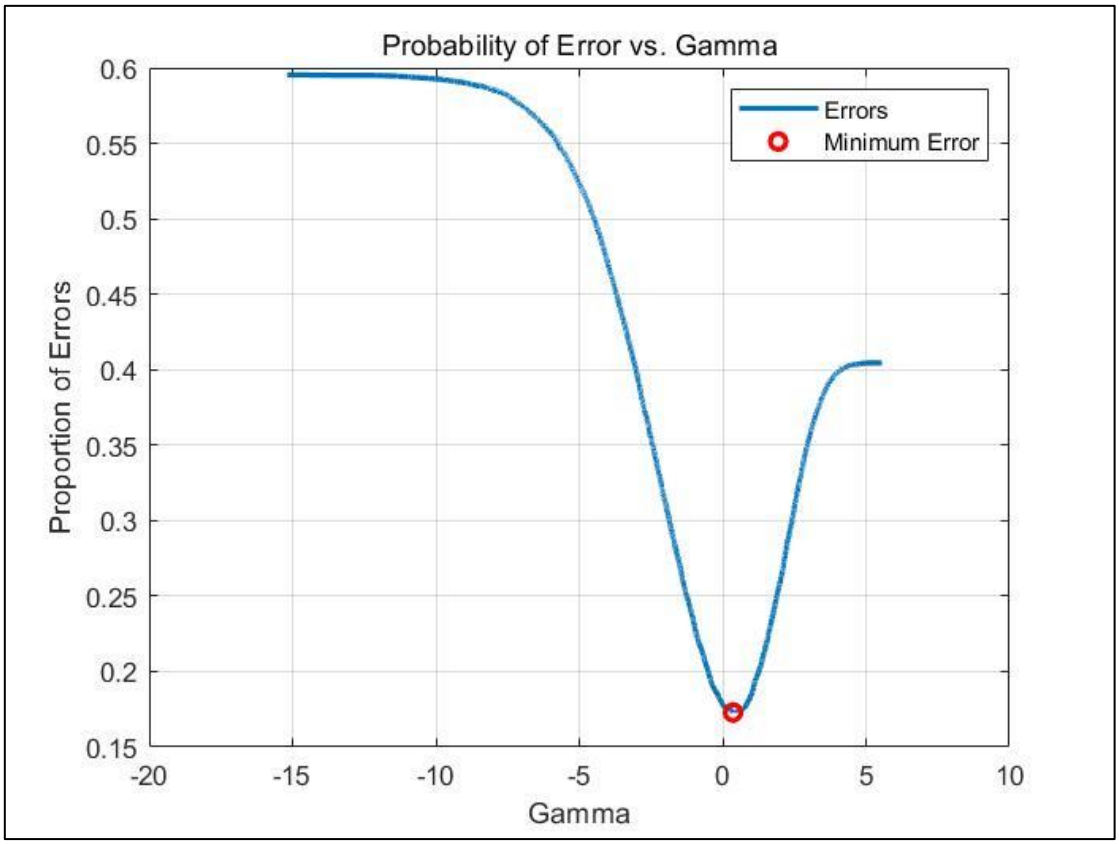

Figure 3: Probability of Error Curve for Ideal Classification

Figure 4 shows the decision space for each distribution along with equal level contours of the discriminant function.

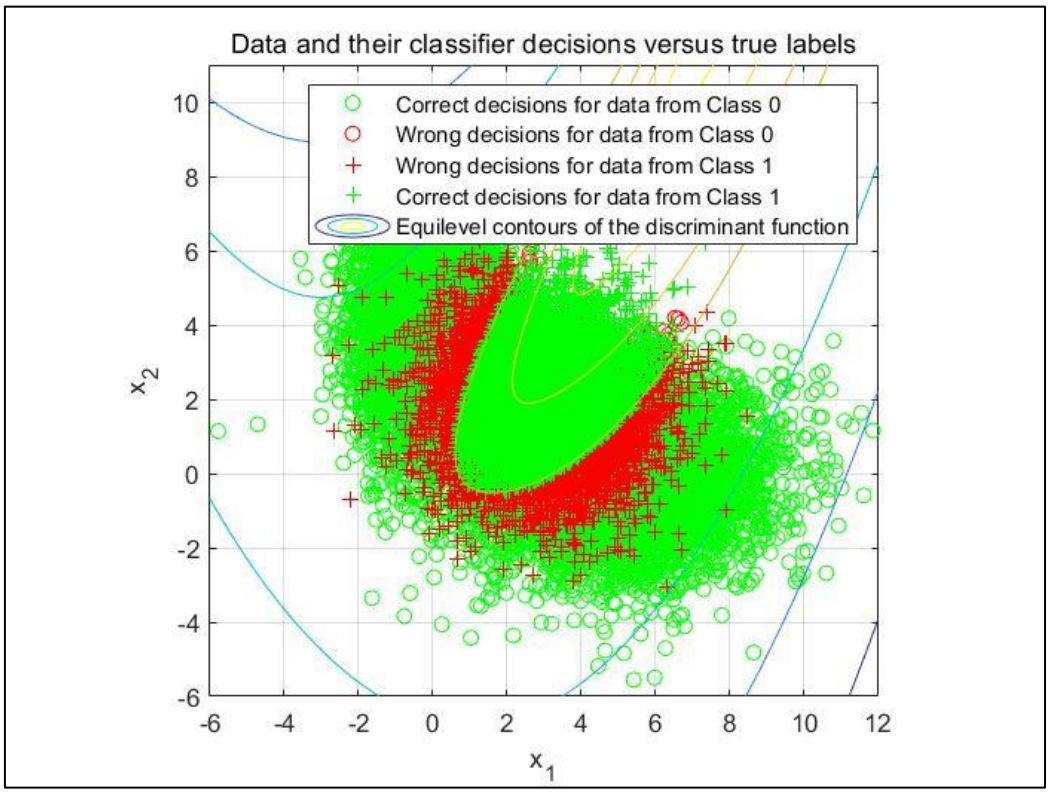

Figure 4: Decision Boundary of Ideal Classifier

Part 2

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. 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.

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

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

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

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

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

Parameter estimation for Class 1 was also performed using 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.

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

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

Table 2, Table 3, and Table 4 show the parameter estimates obtained from analysis of each of the 3 sets of training data. Plots showing the resulting estimated distributions for the Class 0 GMM are shown in Figure 5, Figure 6, and Figure 7.

Table 2: Estimated Means

	D_{train}^{100}	D_{train}^{1000}	D_{train}^{10k}
$\hat{m}_{01}$	$\begin{bmatrix}4.93\\0.05\end{bmatrix}$	$\begin{bmatrix}3.90\\0.05\end{bmatrix}$	$\begin{bmatrix}3.99\\-0.01\end{bmatrix}$
$\hat{m}_{02}$	$\begin{bmatrix}-0.08\\4.73\end{bmatrix}$	$\begin{bmatrix}0.00\\5.05\end{bmatrix}$	$\begin{bmatrix}-0.01\\4.99\end{bmatrix}$
$\hat{m}_1$	$\begin{bmatrix}2.66\\1.73\end{bmatrix}$	$\begin{bmatrix}2.98\\2.09\end{bmatrix}$	$\begin{bmatrix}2.99\\1.97\end{bmatrix}$

Table 3: Estimated Covariances

	D_{train}^{100}	D_{train}^{1000}	D_{train}^{10k}
$\hat{C}_{01}$	$\begin{bmatrix}4.50 & -0.59\\-0.59 & 2.82\end{bmatrix}$	$\begin{bmatrix}4.50 & -0.59\\-0.59 & 2.82\end{bmatrix}$	$\begin{bmatrix}3.90 & 0.02\\0.02 & 1.99\end{bmatrix}$
$\hat{C}_{02}$	$\begin{bmatrix}0.63 & -0.06\\-0.06 & 2.55\end{bmatrix}$	$\begin{bmatrix}0.63 & -0.06\\-0.06 & 2.55\end{bmatrix}$	$\begin{bmatrix}1.03 & 0.01\\0.01 & 2.96\end{bmatrix}$
$\hat{C}_1$	$\begin{bmatrix}1.79 & -0.01\\-0.01 & 1.42\end{bmatrix}$	$\begin{bmatrix}1.92 & 0.06\\0.06 & 2.03\end{bmatrix}$	$\begin{bmatrix}2.01 & 0.00\\0.00 & 2.01\end{bmatrix}$

Table 4: Estimated Alphas for Class 0 GMM

	D_{train}^{100}	D_{train}^{1000}	D_{train}^{10k}
$\widehat{\alpha}_{01}$	0.51	0.53	0.49
$\widehat{\alpha}_{02}$	0.49	0.47	0.51

Table 5: Sample Class Priors

	D_{train}^{100}	D_{train}^{1000}	D_{train}^{10k}
$\hat{P}(L=0)$	0.56	0.59	0.61
$\hat{P}(L=1)$	0.44	0.41	0.39

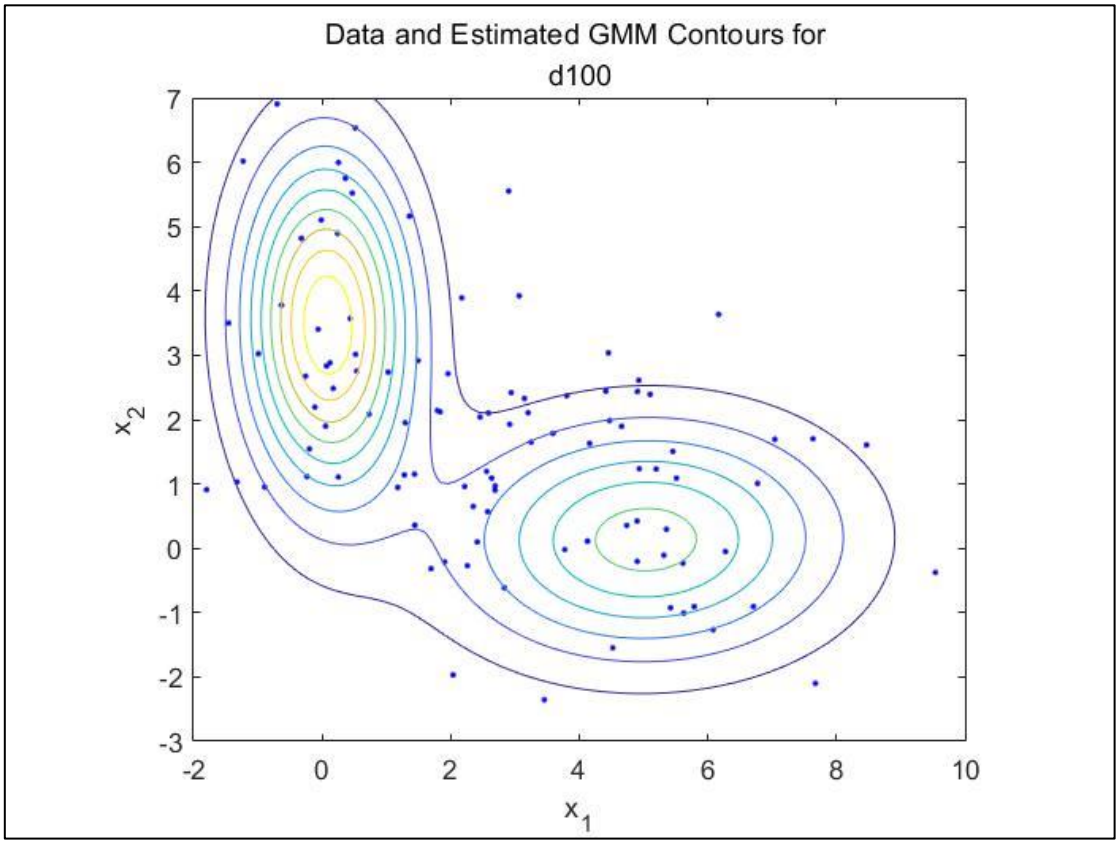

Figure 5: Contour Plot of Estimated Distributions for Class 0 GMM for D100 Training Data

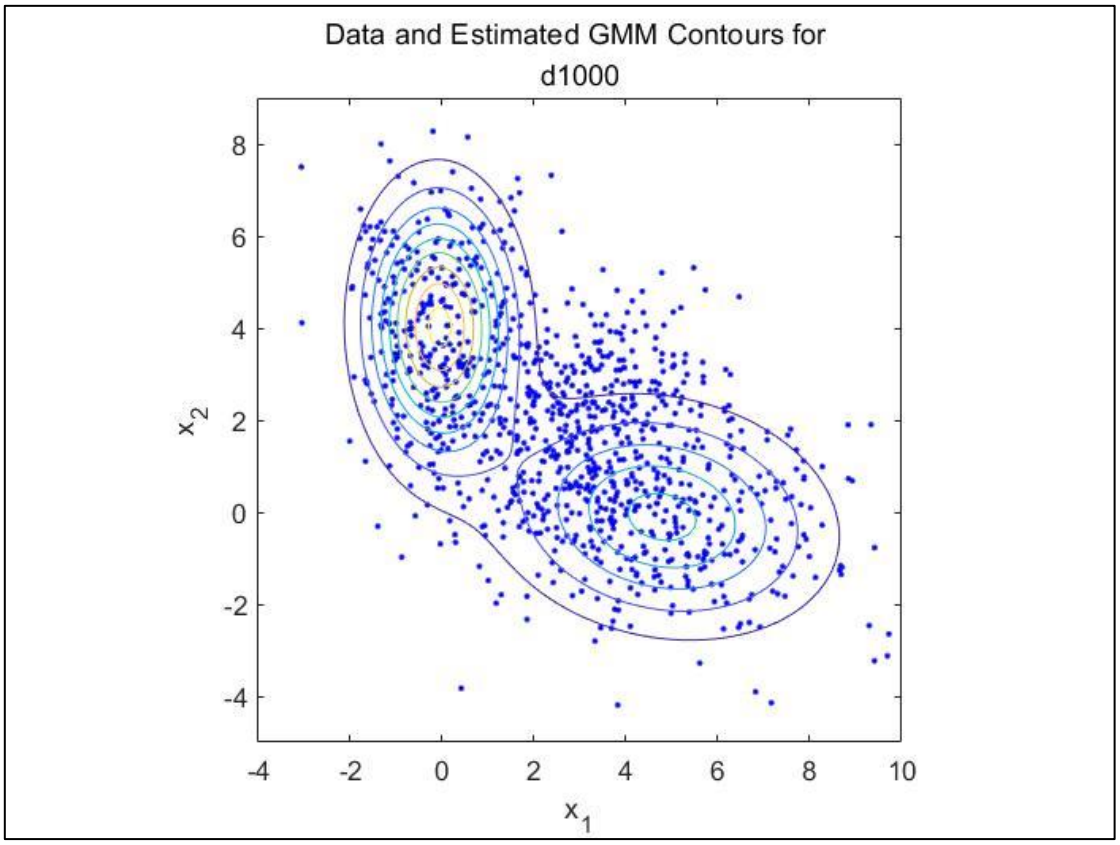

Figure 6: Contour Plot of Estimated Distributions for Class 0 GMM for D1000 Training Data

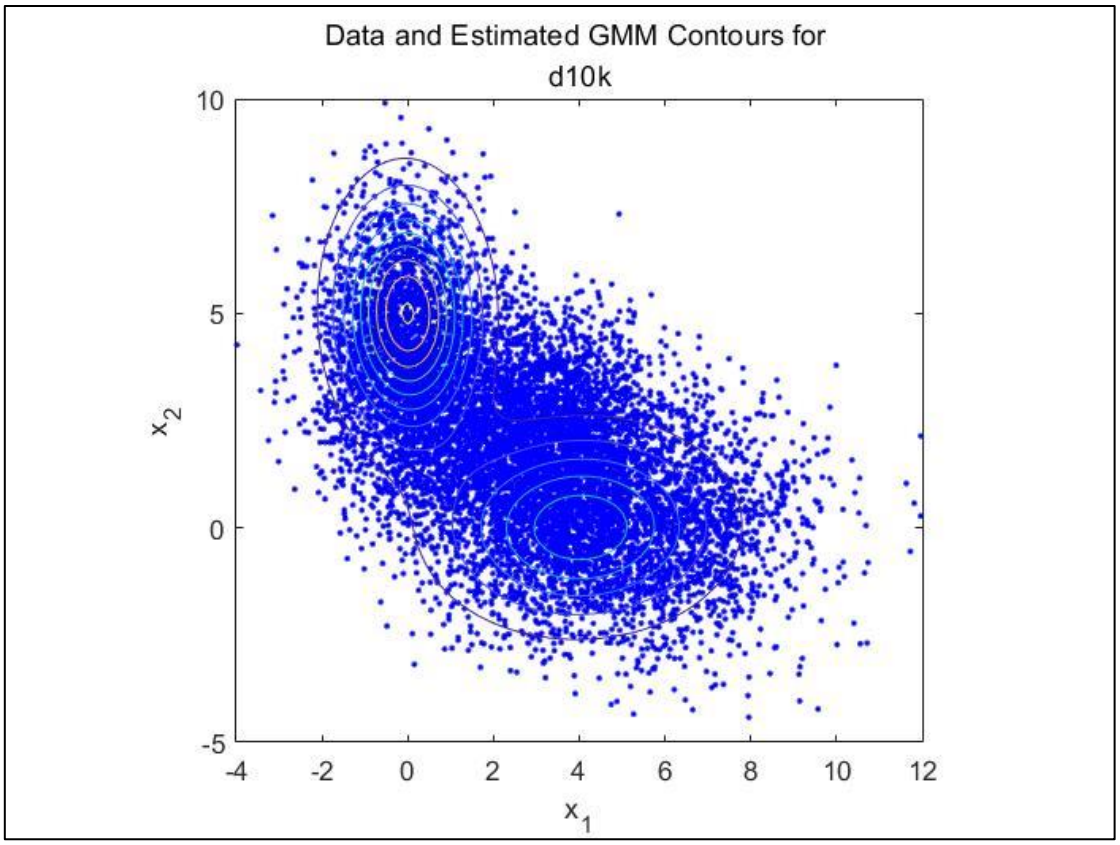

Figure 7: Contour Plot of Estimated Distributions for Class 0 GMM for D10K Training Data

A summary of the minimum estimated probability of errors associated with the parameters estimated from the three training datasets is shown in Table 6. The ROC curves generated by classifying data from the validation data set based on parameters estimated from each of the

training data sets is shown in Figure 8. The minimum probability of error decrease as the number of training samples increases to the point where for the 1000-point training set the probability of error is comparable to the ideal case from part 1.

Table 6: Minimum Probability of Error for all 3 Training Datasets

Training Samples	λ_{minErr}	min P_e
100	1.53	0.1729
1000	1.27	0.1742
10K	1.63	0.1731
Known PDF(Part 1)	1.50	0.1746

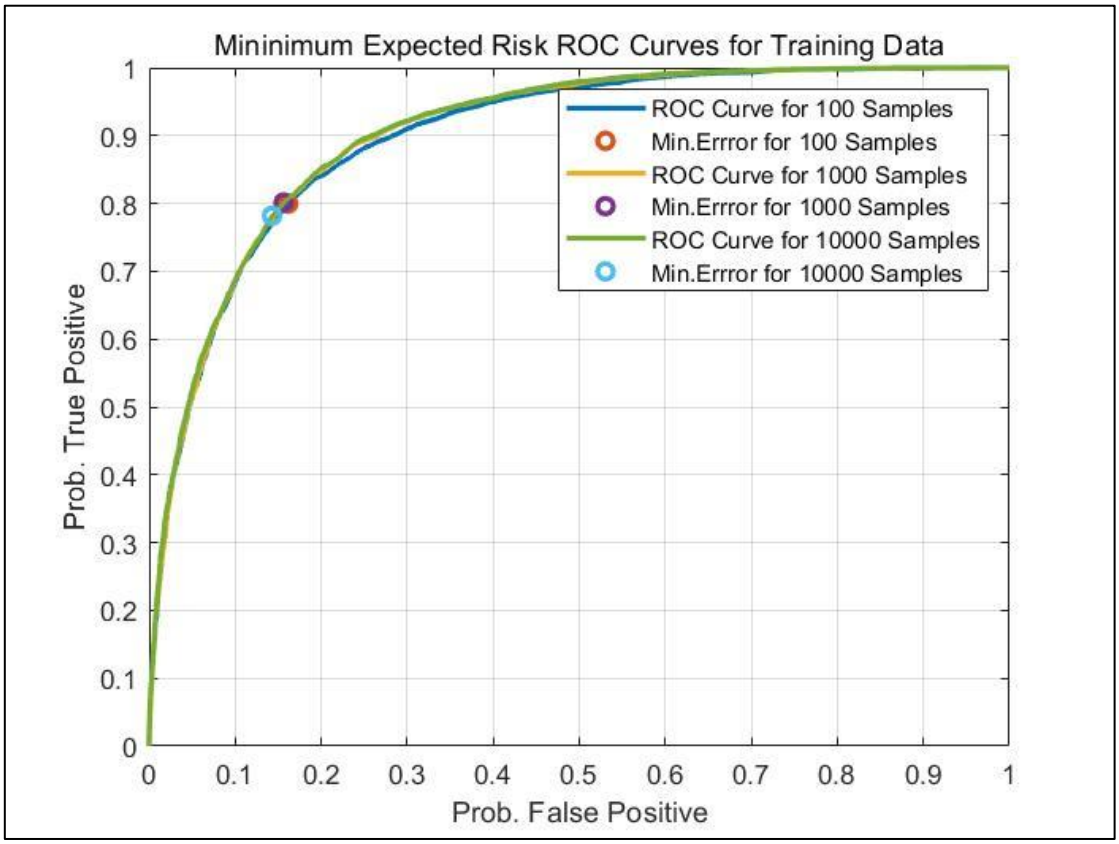

Figure 8: ROC Curves for All Sets of Training Data

Part 3

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 $z(x) = [1 \quad x_1 \quad x_2]^T$

For the quadratic logistic function $z(x) = [1 \quad x_1 \quad x_2 \quad x_1^2 \quad x_1 * x_2 \quad x_2^2]^T$

The w vectors are estimated using numerical optimization techniques with the cost function

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

The minimum expected risk classification criteria are then

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

Table 7 contains a summary of the resulting probability of errors from classifying the 10000 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 1000 sample training data set even approached the theoretical optimal probability of error of 0.174 obtained in Part 1.

Table 7: Logistic Functions Probabilities of Error

Samples	Linear	Quadratic
100	0.437	0.176
1000	0.460	0.174
10000	0.434	0.178

Figure 9, Figure 10, Figure 11, Figure 12, Figure 13, and Figure 14 show the data points of X with classifier decisions and true labels marked.

Figure 9: Classifier for Linear Logistic Fit on D100

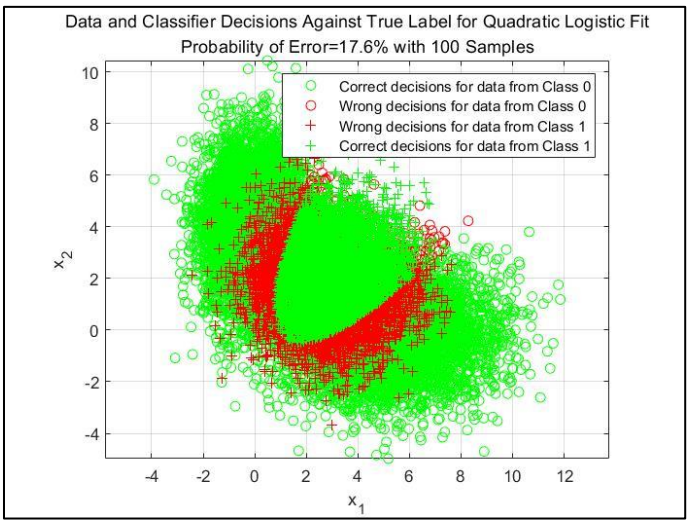

Figure 10: Classifier for Quadratic Logistic Fit on D100

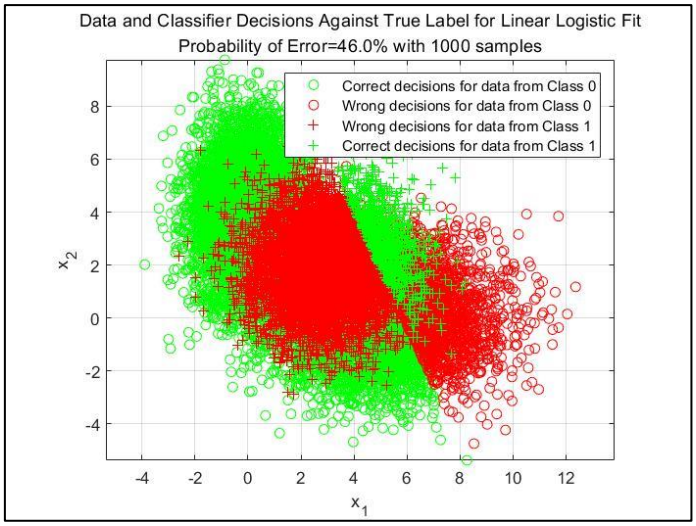

Figure 11: Classifier for Linear Logistic Fit on D1000

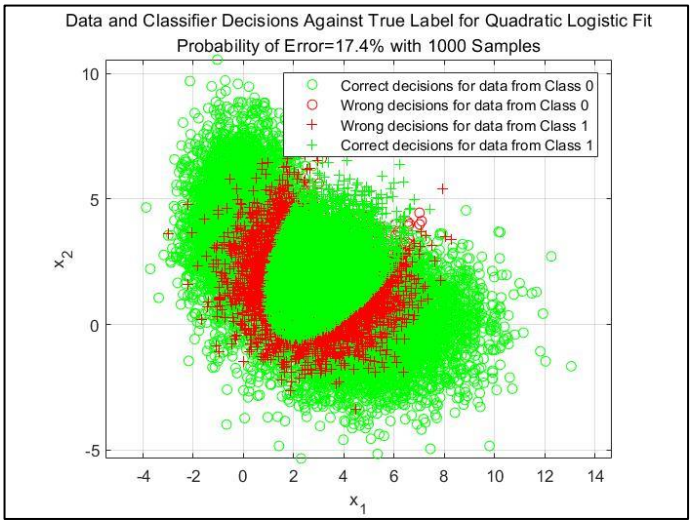

Figure 12: Classifier for Quadratic Logistic Fit on D1000

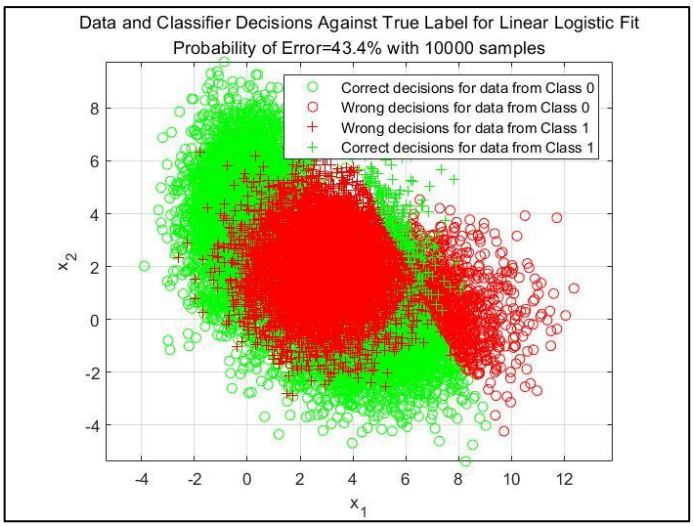

Figure 13: Classifier for Linear Logistic Fit on D10k

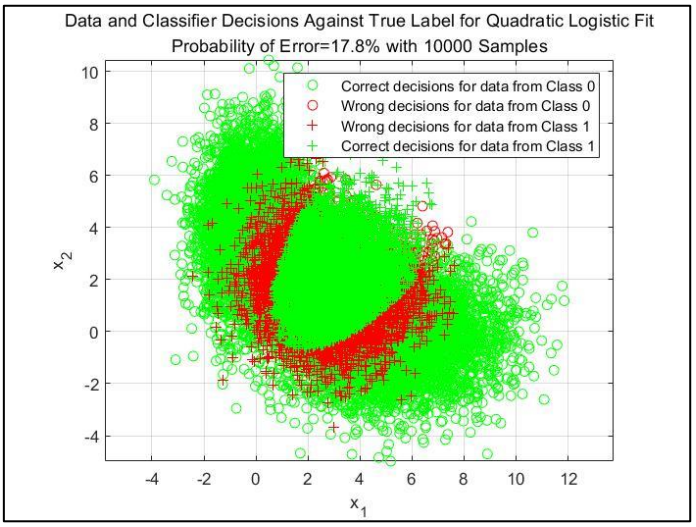

Figure 14 Classifier for Quadratic Logistic Fit on D10k

Question 2

The objective is to find the $[\mathrm{x}, \mathrm{y}]^{\mathrm{T}}$ coordinate position with the highest probability given the prior.

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

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

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

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

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

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

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

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

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

Figure 15, figure 16, figure 17 and figure 18 were created using σx = σy= 0.25 and σi = 0.3.

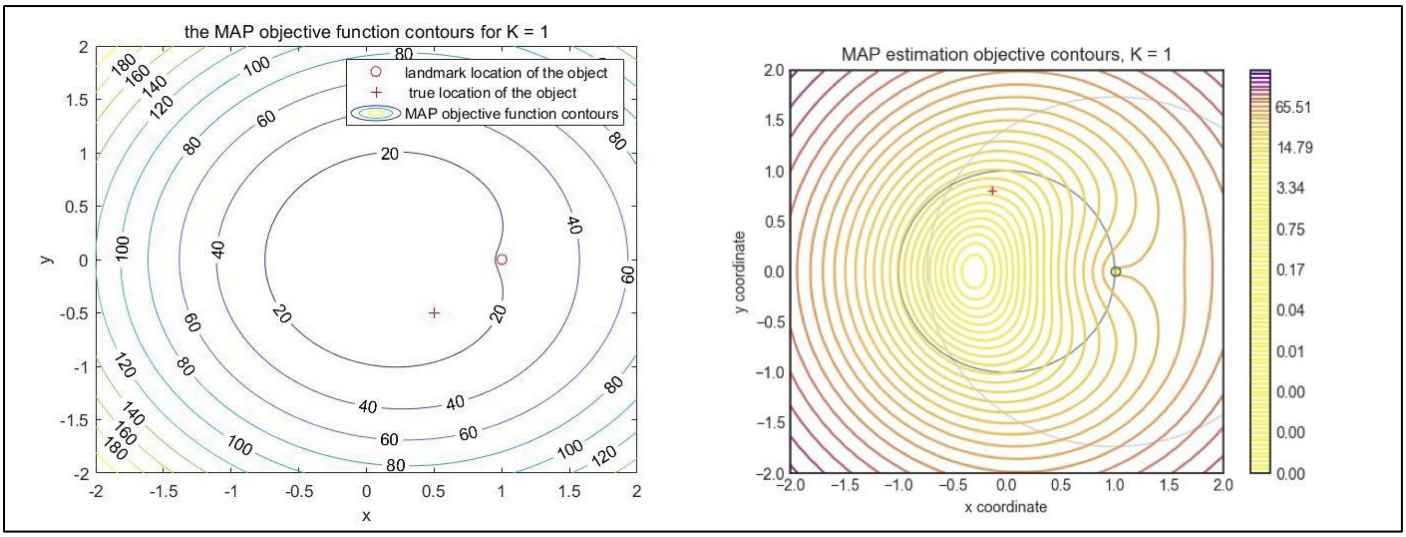

Figure 15: The MAP objective contours for K=1

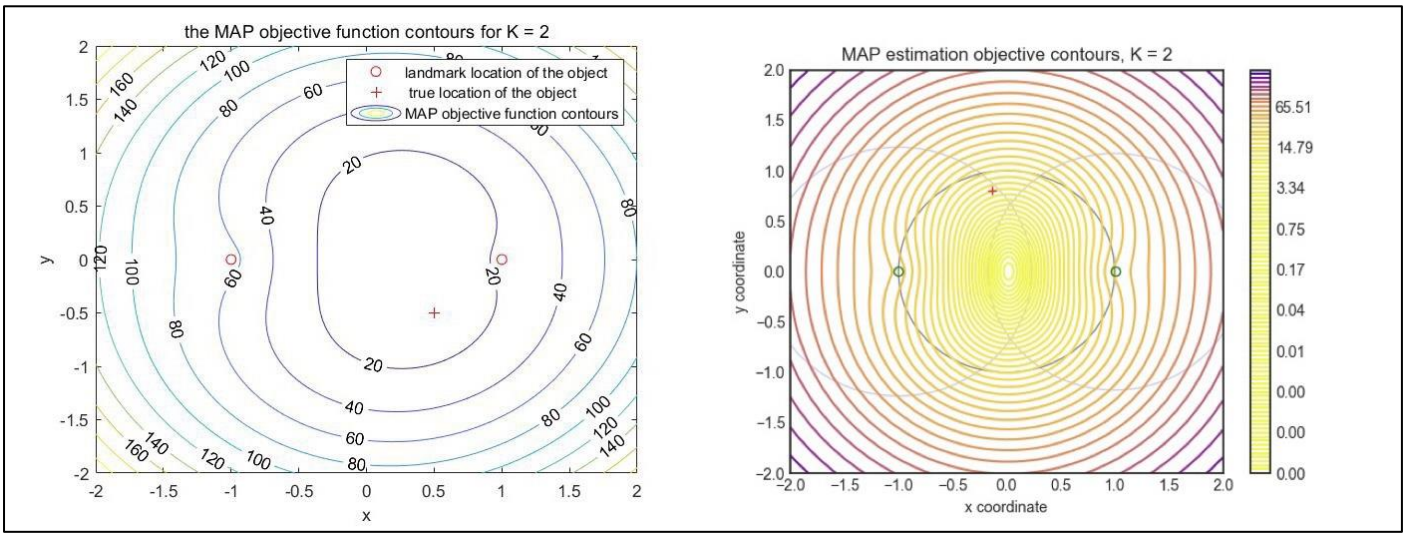

Figure 16: The MAP objective contours for K=2

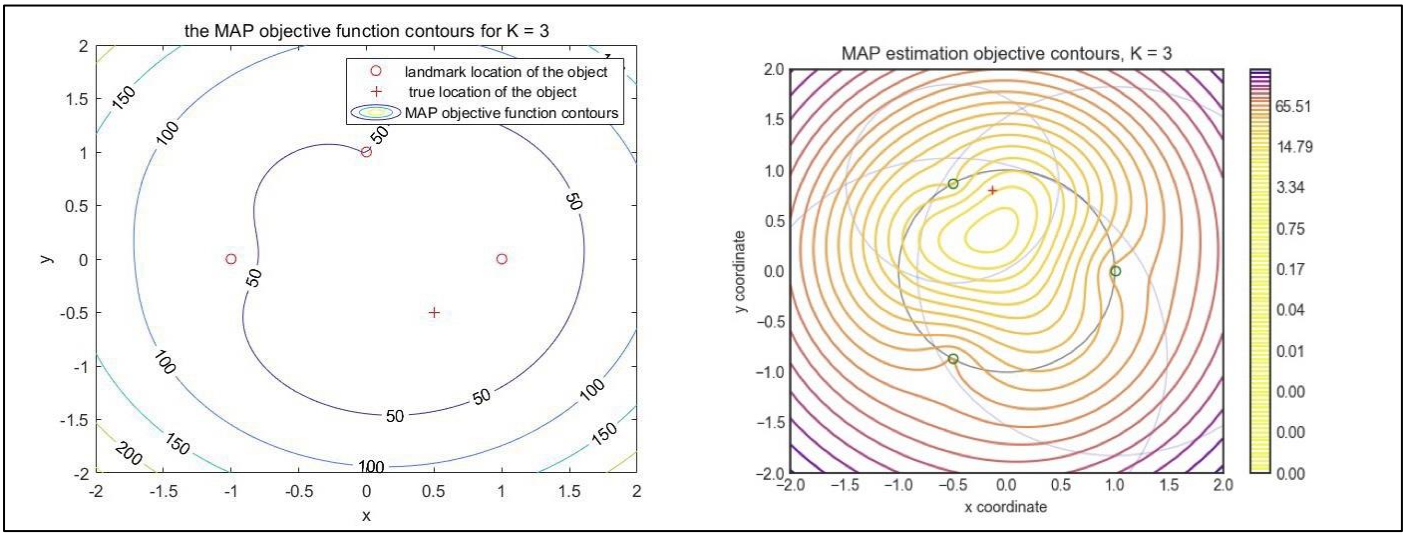

Figure 17: The MAP objective contours for K=3

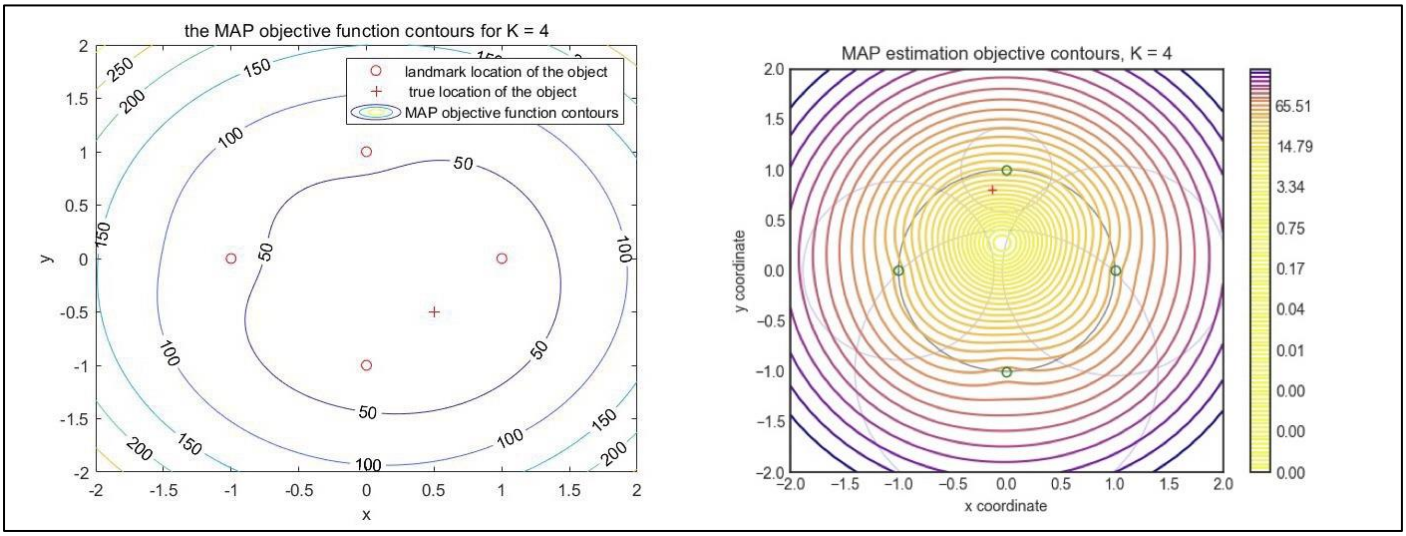

Figure 18: The MAP objective contours for K=4

The code first initializes the variables, then creates an equal level contour in [-2,2], next generates the vehicle location in the presence of Gaussian noise, and marks the correct vehicle location with a red '+ ' to mark the correct vehicle location with a red '+'. The landmark locations are marked with circles.

In the case of K<3, the MAP estimation of the location is not very accurate because the y-coordinate of all landmarks and a priori deviations is 0, while the true location is only within two and one level from the center estimated contour, respectively. In general, MAP estimation becomes more accurate as K increases. However, there are exceptions, for example, in the transition from K = 1 → 2, the range measurement of one of the landmarks happens to be lower than the more accurate estimate of the second landmark, thus causing a measurement error, but as K increases (to about 40), the MAP becomes more accurate.

Question 3

If we choose the category ω max that has the maximum posterior probability, our risk at a point x is:

$$\lambda_s \sum_{j \neq max} P(\omega_j \mid \mathrm{x}) = \lambda_s[1 - P(\omega_{max} \mid \mathrm{x})]$$

whereas if we reject, our risk is λ r. If we choose a non-maximal category ω k (where k _= max), then our risk is

$$\lambda_s \sum_{j \neq k} P(\omega_j \mid \mathrm{x}) = \lambda_s[1 - P(\omega_k \mid \mathrm{x})] \geq \lambda_s[1 - P(\omega_{max} \mid \mathrm{x})]$$

This last inequality shows that we should never decide on a category other than the one that has the maximum posterior probability, as we know from our Bayes analysis. Consequently, we should either choose ω max or we should reject, depending upon which is smaller: $\lambda_s[1 - P(\omega_{max} \mid \mathrm{x})]$ or λ_r. We reject if $\lambda_r \leqslant \lambda_s[1 - P(\omega_{max} \mid \mathrm{x})]$, that is, if $P(\omega_{max} \mid x) \geq (1 - \frac{\lambda_r}{\lambda_s})$

To solve this problem, first, we give the notations that:

$$i, j \in (1, c)$$
$$when\ i = j, \lambda_{ij} = 0$$
$$when\ i = c + 1, \lambda_{ij} = \lambda_r$$
$$else, \lambda_{ij} = \lambda_s$$

To minimize the risk, we need to compute $R(D = i \mid x) = \sum_{j=1}^{c} \lambda_{ij} P(L = j \mid x)$.

$R(D = i \mid x)$ is the cost associated with deciding i, λ_{ij} is the cost(i,j), and $P(L = j \mid x)$is the posterior of j. $\frac{decision}{\text{action}} = \arg \min_i R(D = i \mid x)$

For this problem, consider classes i and k:
risk associated in deciding class i

$$R(D = i \mid x) = \sum_{j=1}^{c} \lambda_{ij} P(L = j \mid x)$$

$$= \sum_{j \neq 1}^{j \leftarrow class\ labels} P(L = j \mid x) \quad \lambda_{ij} = 0 \text{ when } j = i \ \textcircled{1}$$

So $\sum_{j=1}^{C} P(L = j \mid x) = 1$

and then $\sum_{j \neq 1}^{j \leftarrow class\ labels} P(L = j \mid x) = 1 - P[L = i \mid x]$

$\therefore \textcircled{1} \rightarrow R(D = i \mid x) = \lambda_s[1 - P[L = i \mid x]] \ \textcircled{2}$

risk associated in deciding class k is:
$R(D = k \mid x) = \lambda_s[1 - P(L = k \mid x)] \ \textcircled{3}$

To decide between class i and class k:
We can decide class i if and only if $R(D = i \mid x) < R(D = K \mid x)$

that is $R(D = i \mid k) \gtrless_{\mathrm{D=i}}^{\mathrm{D=k}} R(D = k \mid x)$
With equation ② and ③, we have
$\lambda_s[1 - P(L = i \mid x)] \gtrless_{\mathrm{D=i}}^{\mathrm{D=k}} \lambda_s[1 - P(L = k \mid x)]$
$P(L = i \mid x) \gtrless_{\mathrm{P=k}}^{\mathrm{P=i}} P(L = k \mid x)$

Then decide class k if and only if $P(L = i \mid x) \geq P(L = K \mid x)$ for all class k = (1,c) and k ≠i.
And we still need to decide between class i and reject class(c+1).
Risk associated with reject class is:

$$R[D = c + 1 \mid x] = \sum_{j=1}^{C} \lambda_{ij} P(L = j \mid x) = \lambda_{\mathrm{r}} \sum_{j=1}^{C} P(L = \dot{\gamma} \mid x) \quad ④$$

With equation ②, we have $R(D = i \mid x) = \lambda_S[1 - P(L = i \mid x)]$

And minimize risk associated with decision, we get decide class i if and only if

$$R(D = i \mid x) \leq R(D = c + 1 \mid x)$$

$$\lambda_S[1 - P(L = i \mid x)] \leq \lambda_R$$

$$1 - P(L = i \mid x) \leq \frac{\lambda_R}{\lambda_S}$$

$$P(L = i \mid x) \geq (1 - \frac{\lambda_R}{\lambda_S}) \quad ⑤$$

If $P(L = i \mid x) \geq \left(1 - \frac{\lambda_R}{\lambda_S}\right)$ choose reject class c+1, we have:

When $\lambda_{\mathrm{r}} = 0$, consider equation ⑤, we have $P(L = i \mid x) \geq (1 - \frac{0}{\lambda_S})$, that is $P(L = i \mid x) \geq 1$. This means decide class i only if posterior is larger or equal to (⩾)1. This is almost impossible since if $\lambda_{\mathrm{r}} = 0$, cost associated with reject class is also 0. Therefore, decision rule will always classify to the reject class.

When $\lambda_{\mathrm{r}} > \lambda_{\mathrm{s}}$ and $\lambda_{\mathrm{r}} \neq 0$, consider equation ⑤,we have $P(L = i \mid x) \geq (1 - \frac{\lambda_r}{\lambda_S})$, the right hand side of this equation is smaller than 0. So $P(L = i \mid x) \geq \left(1 - \frac{\lambda_r}{\lambda_S}\right)$ and $\left(1 - \frac{\lambda_r}{\lambda_S}\right) > 0$ are true for all posterior $P(x) > 0$.And then decision rule will always classify to the same class i and never classify to the reject class. This is intuitive with a high cost of reject class, so we'd better minimize the risk.

Appendix code for Question 1

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%EECE5644 Fall 2021
% Wang Yinan 001530926 | HW2
%%=========================Question 1=========================%%
% Code help and example from Prof.Deniz
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all;close all;clc;

%%=========================Setup=========================%%
dimension=2; %Dimension of data
%Define data
D.d100.N=100;
D.d1000.N=1000;
D.d10k.N=10000;
D.d20k.N=20e3;
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(1);
%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','Class 1');
    grid on;
    xlabel('x1');ylabel('x2');
    title([num2str(D.(dTypes{ind}).N) ' Samples From Two Classes']);
end
legend 'show';

%%=========================Part 1=========================%%
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,p10_ideal,p11_ideal);
plotMinPFE(logGamma,prob.pFE,prob.min_pFE_ind);
plotDecisions(D.d20k.x,D.d20k.labels,decision_ideal);
plotERMContours(D.d20k.x,alpha0,mu0,Sigma0,mu1,Sigma1,logGamma_ideal);
fprintf('Theoretical: Gamma=%1.2f, Error=%1.2f%%\n',...
    exp(logGamma_ideal),100*pFE_ideal);
fprintf('Estimated: Gamma=%1.2f, Error=%1.2f%%\n',minGAMMA,100*prob.min_pFE);

%%=========================Part 2=========================%%
roc=zeros(4,20001,3);
samples=[100 1000 10000 20000];
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,dTypes{ind});
    %Calculate discriminate score
    px0=pdf(D.(dTypes{ind}).DMM_Est0,D.d20k.x');
    px1=pdf(D.(dTypes{ind}).DMM_Est1,D.d20k.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
    fprintf('Estimated: Gamma=%1.2f, Error=%1.2f%%\n',...
        minGAMMA,100*prob.min_pFE);
    roc(1,:,ind)=prob.p10;
    roc(2,:,ind)=prob.p11;
    roc(3,:,ind)=prob.min_FP;
    roc(4,:,ind)=prob.min_TP;
end
figure;
for ind=1:length(dTypes)-1
    nameR=('ROC Curve for '+string(samples(ind))+' Samples');
    nameM=('Min.Errror for '+string(samples(ind))+' Samples');
    plot(roc(1,:,ind),roc(2,:,ind),'DisplayName',nameR,'LineWidth',2);
    hold on;
    plot(roc(3,:,ind),roc(4,:,ind),'o','DisplayName',nameM,'LineWidth',2);
    hold on;
end
xlabel('Prob. False Positive');
ylabel('Prob. True Positive');
title('Mininimum Expected Risk ROC Curves for Training Data');
legend 'show';
```

```
grid on; box on;

%%=========================Part 3=========================%%
options=optimset('MaxFunEvals',3000,'MaxIter',10000);
for ind=1:length(dTypes)-1
    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}).labels)),...
        lin.init,options);
    lin.discScore=lin.theta'*[ones(1,D.d20k.N); D.d20k.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.d20k.N,1) D.d20k.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%% ' ...
        'with %s samples'], ...
        100*lin.prob.pFE,string(samples(ind))));
    % plotDecisions(D.d20k.x,D.d20k.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%% ' ...
        'with %d Samples'], ...
        100*quad.prob.pFE,samples(ind)));
end

%%=========================Question 1 Functions=========================%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Functions credit to Prof.Deniz
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
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,data)
figure
if size(x,1)==2
plot(x(1,:),x(2,:),'b.');
xlabel('x_1'), ylabel('x_2'), title('Data and Estimated GMM Contours for ',data),
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,p10_ideal,p11_ideal)
figure;
plot(p10,p11,'DisplayName','ROC Curve','LineWidth',2);
hold all;
plot(min_FP,min_TP,'o','DisplayName','Estimated Min. Error','LineWidth',2);
hold all;
plot(p10_ideal,p11_ideal,'+','DisplayName','Ideal Min. Error');
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);
%plot(min_FP,min_TP,'+','DisplayName','Calculated Min. Error');
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
plot(x(1,ind00),x(2,ind00),'og','DisplayName','Class 0, Correct'); hold on,
plot(x(1,ind10),x(2,ind10),'or','DisplayName','Class 0, Incorrect'); hold on,
plot(x(1,ind01),x(2,ind01),'+r','DisplayName','Class 1, Correct'); hold on,
plot(x(1,ind11),x(2,ind11),'+g','DisplayName','Class 1, Incorrect'); hold on,
axis equal,
grid on;
title('Data and their classifier decisions versus true labels');
xlabel('x_1'), ylabel('x_2');
legend('Correct decisions for data from Class 0',...
'Wrong decisions for data from Class 0',...
'Wrong decisions for data from Class 1',...
'Correct decisions for data from Class 1');
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function plotERMContours(x,alpha0,mu0,Sigma0,mu1,Sigma1,logGamma_ideal)
horizontalGrid = linspace(floor(min(x(1,:))),ceil(max(x(1,:))),101);
verticalGrid = linspace(floor(min(x(2,:))),ceil(max(x(2,:))),91);
[h,v] = meshgrid(horizontalGrid,verticalGrid);
discriminantScoreGridValues =...
log(evalGaussian([h(:)';v(:)'],mu1,Sigma1))-log(evalGMM([h(:)';v(:)'],...
alpha0,mu0,Sigma0)) - logGamma_ideal;
minDSGV = min(discriminantScoreGridValues);
maxDSGV = max(discriminantScoreGridValues);
discriminantScoreGrid = reshape(discriminantScoreGridValues,91,101);
contour(horizontalGrid,verticalGrid,...
discriminantScoreGrid,[minDSGV*[0.9,0.6,0.3],0,[0.3,0.6,0.9]*maxDSGV]); % plot equilevel contours of the
discriminant function
% including the contour at level 0 which is the decision boundary
legend('Correct decisions for data from Class 0',...
'Wrong decisions for data from Class 0',...
'Wrong decisions for data from Class 1',...
'Correct decisions for data from Class 1',...
'Equilevel contours of the discriminant function' ),
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Appendix code for Question 2

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%EECE5644 Fall 2021
% Wang Yinan 001530926 | HW2
%%=========================Question 2=========================%%
% Code help and example from Prof.Deniz
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all;close all;clc;

%%=========================Setup=========================%%
% variances
xT = 0.5;
yT = -0.5;
xi = [1, -1, 0, 0];
yi = [0, 0, 1, -1];
SIGMA_X = 0.25;
SIGMA_Y = 0.25;
SIGMA_I = 0.3;
ni = normrnd(0, SIGMA_I^2, 4, 1);
for i = 1:4
    ri(i) = sqrt((xT-xi(i))^2+(yT-yi(i))^2)+ni(i);
end

% Creates an equilevel contour for the MAP estimation objective function
x = linspace(-2,2);
y = linspace(-2,2);
[X,Y] = meshgrid(x,y);
r2 = zeros(10000,4);
s2 = zeros(10000,4);
gMAP = zeros(10000,4);

% Calculates values of the MAP estimation objective function on a given mesh
for i = 1:4
s1 = X(:).^2/(SIGMA_X)^2+Y(:).^2/(SIGMA_Y)^2;
r2(:,i) = sqrt((X(:)-xi(i)).^2+(Y(:)-yi(i)).^2);
s2(:,i) = ((ri(i)-r2(:,i)).^2)/(SIGMA_I^2);
end

% The MAP objective contours
gMAP(:,1) = s1+s2(:,1);
gMAP(:,2) = s1+s2(:,1)+s2(:,2);
gMAP(:,3) = s1+s2(:,1)+s2(:,2)+s2(:,3);
gMAP(:,4) = s1+s2(:,1)+s2(:,2)+s2(:,3)+s2(:,4);
GMAP1 = reshape(gMAP(:,1),100,100);
GMAP2 = reshape(gMAP(:,2),100,100);
GMAP3 = reshape(gMAP(:,3),100,100);
GMAP4 = reshape(gMAP(:,4),100,100);
```

```
% PLOT
% for K=1
figure(15);
plot(xi(1),yi(1),'or'); hold on,
plot(xT,yT,'+r'); hold on,
contour(X,Y,GMAP1,'ShowText','on');
legend('landmark location of the object',' true location of the object','MAP objective function contour
s'),
title('the MAP objective function contours for K = 1'),
xlabel('x'), ylabel('y')

% for K=2
figure(16);
plot(xi(1:2),yi(1:2),'or'); hold on,
plot(xT,yT,'+r'); hold on,
contour(X,Y,GMAP2,'ShowText','on');
legend('landmark location of the object',' true location of the object','MAP objective function contour
s'),
title('the MAP objective function contours for K = 2'),
xlabel('x'), ylabel('y')

% for K=3
figure(17);
plot(xi(1:3),yi(1:3),'or'); hold on,
plot(xT,yT,'+r'); hold on,
contour(X,Y,GMAP3,'ShowText','on');
legend('landmark location of the object',' true location of the object','MAP objective function contour
s'),
title('the MAP objective function contours for K = 3'),
xlabel('x'), ylabel('y')

% for K=4
figure(18);
plot(xi(1:4),yi(1:4),'or'); hold on,
plot(xT,yT,'+r'); hold on,
contour(X,Y,GMAP4,'ShowText','on');
legend('landmark location of the object',' true location of the object','MAP objective function contour
s'),
title('the MAP objective function contours for K = 4'),
xlabel('x'), ylabel('y')


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%=========================python=========================%%
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 pair within the unit circle centered at the origin.
def random_unit_circle_coords():
    # Polar coordinates with a square-rooted r-value produce a uniform distribution
    r = np.sqrt(np.random.uniform(0, 1))
    theta = np.random.uniform(0, 2) * np.pi
    x = r * np.cos(theta)
    y = r * np.sin(theta)
    return np.array([x, y])

# Generates noisy measurements for all K landmarks given the true position.
def get_range_measurements(K, xy_true):
    return [generate_measurement(landmark_pos(i, K), xy_true) for i in range(K)]

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

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

# Creates an equilevel contour plot for the MAP estimation objective function
# given a set of range measurements
def plot_equilevels(range_measurements, xy_true):
    # First, create 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_objective(gridpoints, range_measurements)

    # Then, set up the 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(range_measurements):
        (x, y) = landmark_pos(i, len(range_measurements))
        plt.plot((x), (y), 'o', color='g', markerfacecolor='none')
        # I added faint blue circles to demonstrate the range 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(range_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_objective(xy, range_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(range_measurements):
        xy_i = landmark_pos(i, len(range_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


xy_true = random_unit_circle_coords()

for K in [1, 2, 3, 4]:
    range_measurements = get_range_measurements(K, xy_true)
    plot_equilevels(range_measurements, xy_true)
```