EECE5644 Assignment2

Name: Liangshe Li
NUID: 001586357

Question1(40%)

The probability density function (pdf) for a 2-dimensional real-valued random vector X is as follows: p(x) = P(L = 0)p(x|L = 0)+P(L = 1)p(x|L = 1). Here L is the true class label that indicates which class-label-conditioned pdf generates the data.

The class priors are P(L = 0) = 0.65 and P(L = 1) = 0.35. The class class-conditional pdfs are p(x|L=0)=w1g(x|m01,C01)+w2g(x|m02,C02) and p(x|L=1)=g(x|m1,C1), where g(x|m,C) is a multivariate Gaussian probability density function with mean vector m and covariance matrix C. The parameters of the class-conditional Gaussian pdfs are: w1 = w2 = 1/2, and

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

For numerical results requested below, generate the following independent datasets each consisting of iid samples from the specified data distribution, and in each dataset make sure to include the true class label for each sample.

- D_{train}^{20} consists of 20 samples and their labels for training;
- D_{train}^{200} consists of 200 samples and their labels for training;
- D_{train}^{2000} consists of 2000 samples and their labels for training;
- $D_{validate}^{10K}$ consists of 10000 samples and their labels for validation;

Part 1: (10%) Determine the theoretically optimal classifier that achieves minimum probability of error using the knowledge of the true pdf. Specify the classifier mathematically and implement it; then apply it to all samples in $D_{validate}^{10K}$. From the decision results and true labels for this validation set, estimate and plot the ROC curve of this min-P(error) classifier, and on the ROC curve indicate, with a special marker, the location of the min-P(error) classifier. Also report an estimate of the min-P(error) achievable, based on counts of decision-truth label pairs on $D_{validate}^{10K}$.

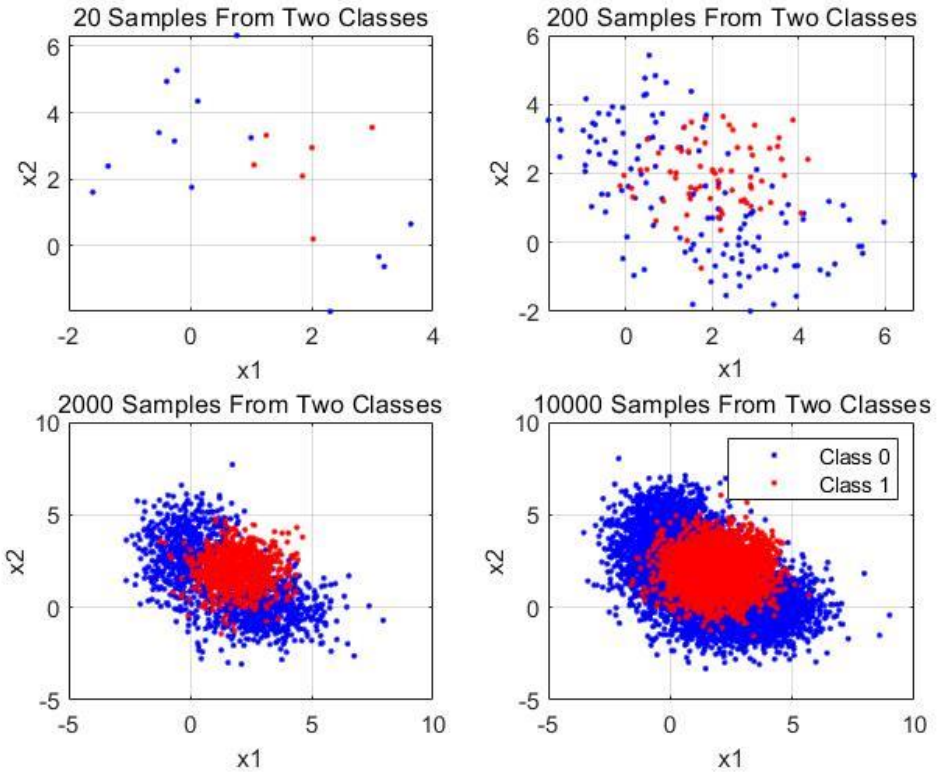
Optional: As supplementary visualization, generate a plot of the decision boundary of this classification rule overlaid on the validation dataset. This establishes an aspirational performance level on this data for the following approximations.

Part 2: (30%) (a) Using the maximum likelihood parameter estimation technique train three separate logistic-linear-function-based approximations of class label posterior functions given a sample. For each approximation use one of the three training datasets D_{train}^{20}, D_{train}^{200}, D_{train}^{2000}. When optimizing the parameters, specify the optimization problem as minimization of the negative-loglikelihood of the training dataset, and use your favorite numerical optimization approach, such as gradient descent or Matlab's fminsearch. Determine how to use these class-label-posterior approximations to classify a sample in order to approximate the minimum-P(error) classification rule; apply these three approximations of the class label posterior function on samples in $D_{validate}^{10K}$, and estimate the probability of error that these three classification rules will attain (using counts of decisions on the validation set). Optional: As supplementary visualization, generate plots of the decision boundaries of these trained

classifiers superimposed on their respective training datasets and the validation dataset.
(b) Repeat the process described in Part (2a) using a logistic-quadratic-function- based approximation of class label posterior functions given a sample. How does the performance of your classifiers trained in this part compare to each other considering differences in number of training samples and function form? How do they compare to the theoretically optimal classifier from Part 1? Briefly discuss results and insights.

Note 1: With $\mathbf{x}$ representing the input sample vector and $\mathbf{w}$ denoting the model parameter vector, logistic-linear-function refers to $h(\mathbf{x},\mathbf{w}) = 1/(1+e^{-\mathbf{w}^{\mathrm{T}}\mathbf{z}(\mathbf{x})})$, where $\mathbf{z}(\mathbf{x}) = [1,\mathbf{x}^{\mathrm{T}}]^{\mathrm{T}}$; and logistic-quadratic-function refers to $h(\mathbf{x},\mathbf{w}) = 1/(1+e^{-\mathbf{w}^{\mathrm{T}}\mathbf{z}(\mathbf{x})})$, where $\mathbf{z}(\mathbf{x}) = [1,x_1,x_2,x_1^2,x_1x_2,x_2^2]^{\mathrm{T}}$.

Answer:
Plots for datasets are shown below:



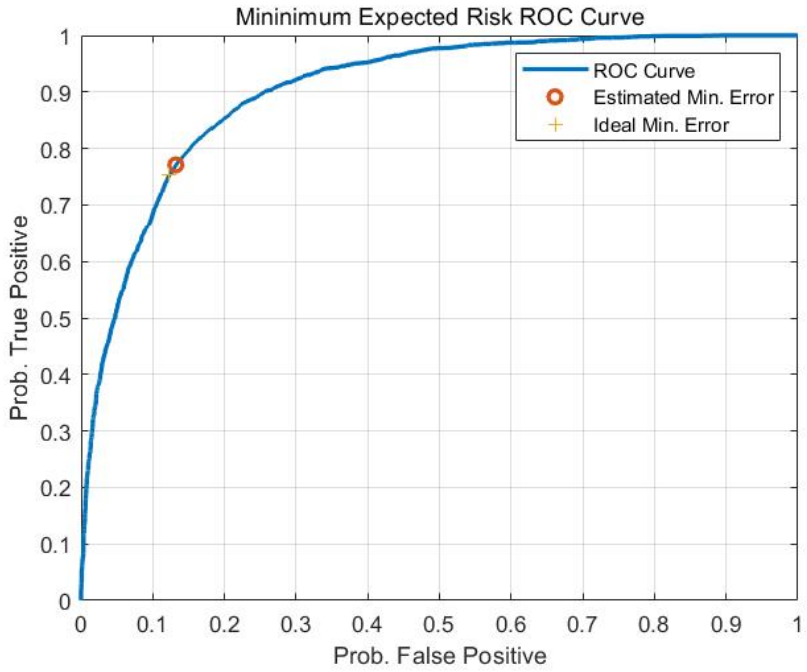
Part 1
The minimum expected risk classification rule:

$$\frac{P(x|L=1)}{P(x|L=0)} \lessgtr \frac{(\lambda_{10}-\lambda_{00})P(L=0)}{(\lambda_{01}-\lambda_{11})P(L=1)} = \gamma$$

That is:

$$(D(x)=1)\frac{P(x|L=1)}{P(x|L=0)} > \frac{(1-0)*0.65}{(1-0)*0.35} = \gamma \approx 1.857$$

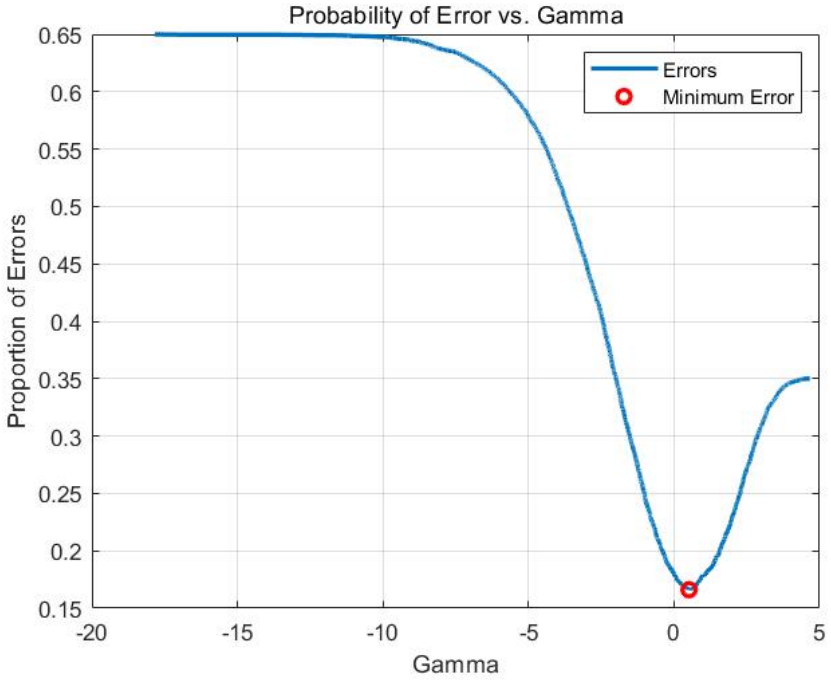
The classifier was implemented for multiple values of gamma and the ROC curve is shown in Figure 2 below. The locations of the theoretical minimum error(orange plus)as well as the minimum error determined by a parametric sweep of gamma(red circle) are marked on the plot.



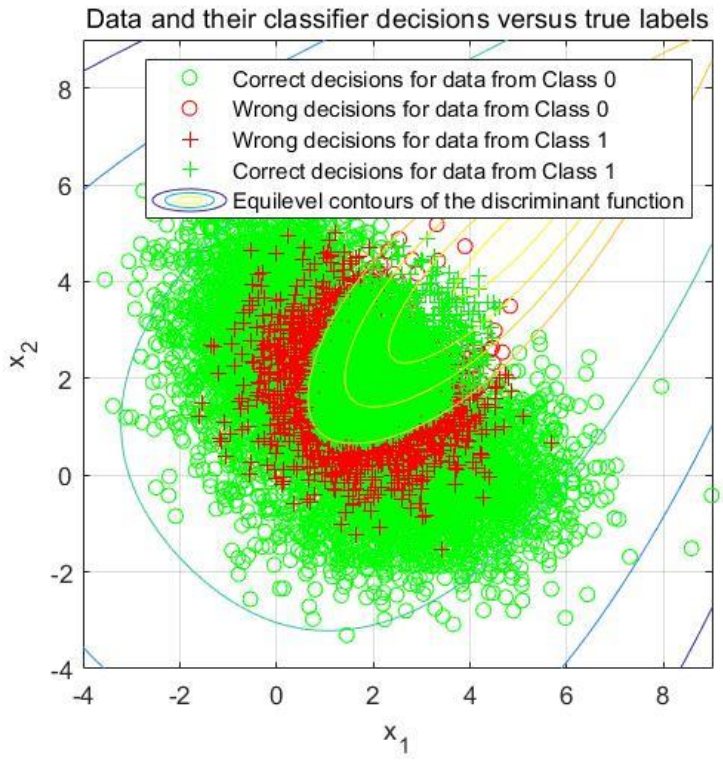
The following table compares the theoretical and the calculated minimum probability of error. As can be seen the two values closely align providing confidence in the estimated value.

	γ	Min P_e
Theoretical	1.86	0.1664
Calculated from data	1.68	0.1661

The probability of error versus Gamma with the calculated and estimated minimum error points marked are shown in the figure below:



The figure below shows the decision space for each distribution along with equal level contours of the discriminant function.



Part 2

(a) In this part maximum likelihood parameter estimation techniques were used to train logistic linear 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 20, 200, and 2000 samples respectively.

Logistic linear function refer to:

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

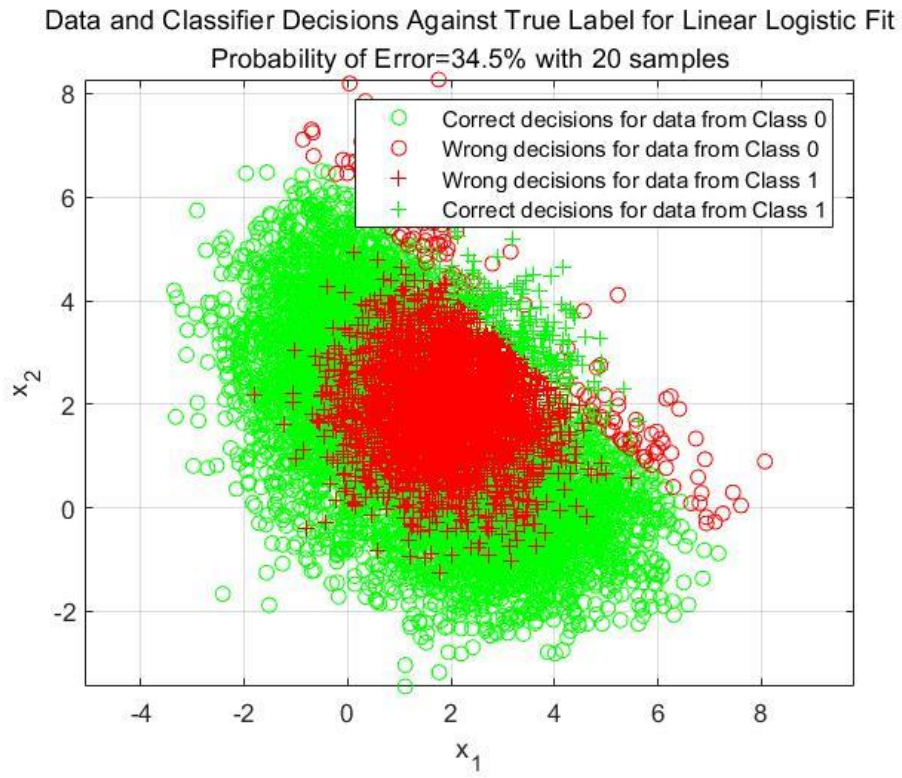
The w vectors are estimated using numerical optimization techniques with the cost function:

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

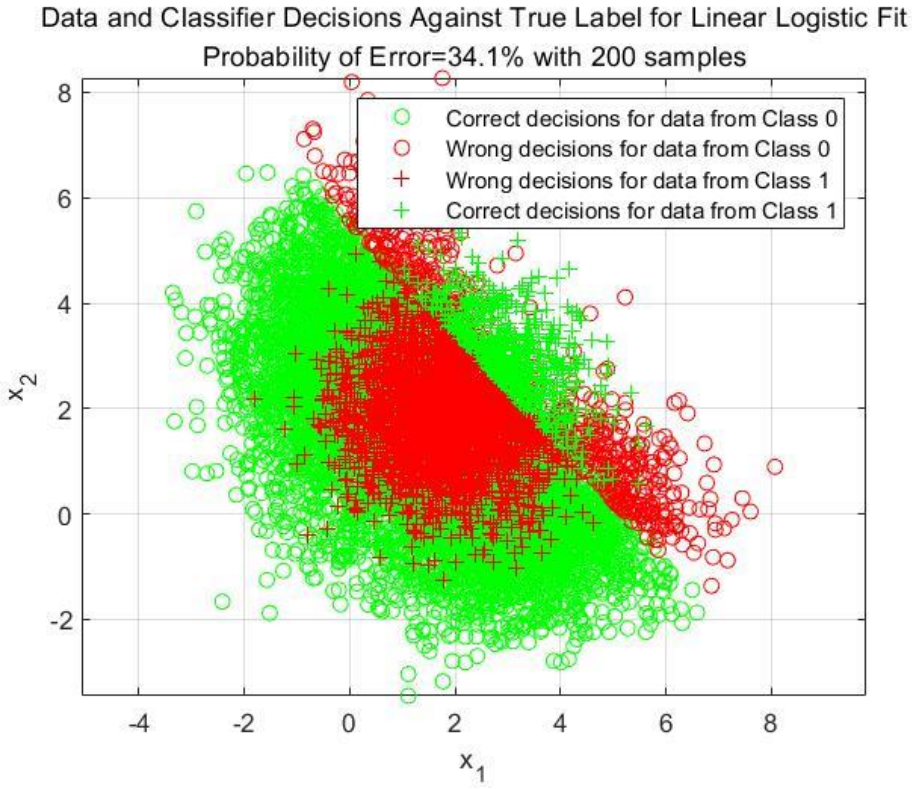
The minimum expected risk classification criteria are then:

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

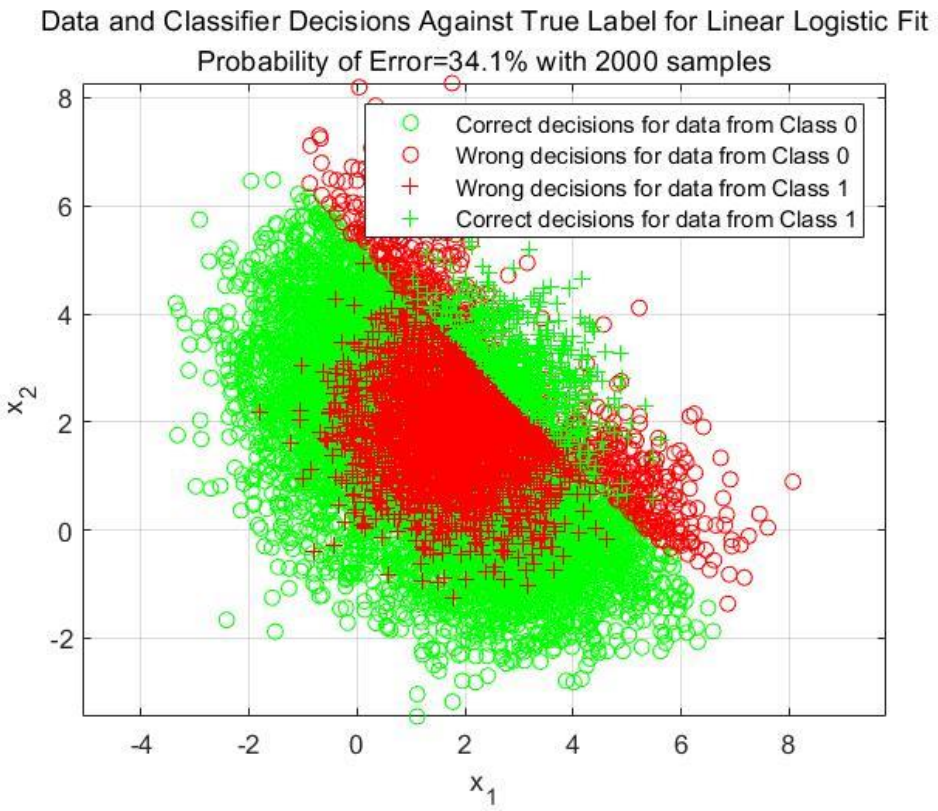
The next plot shows whether classifier decisions (Linear function) are correct or not using 20 samples:



The next plot shows whether classifier decisions(Linear function) are correct or not using 200 samples:



The next plot shows whether classifier decisions (Linear function) are correct or not using 2000 samples:



The result of probability of error using linear function is shown in the below table:

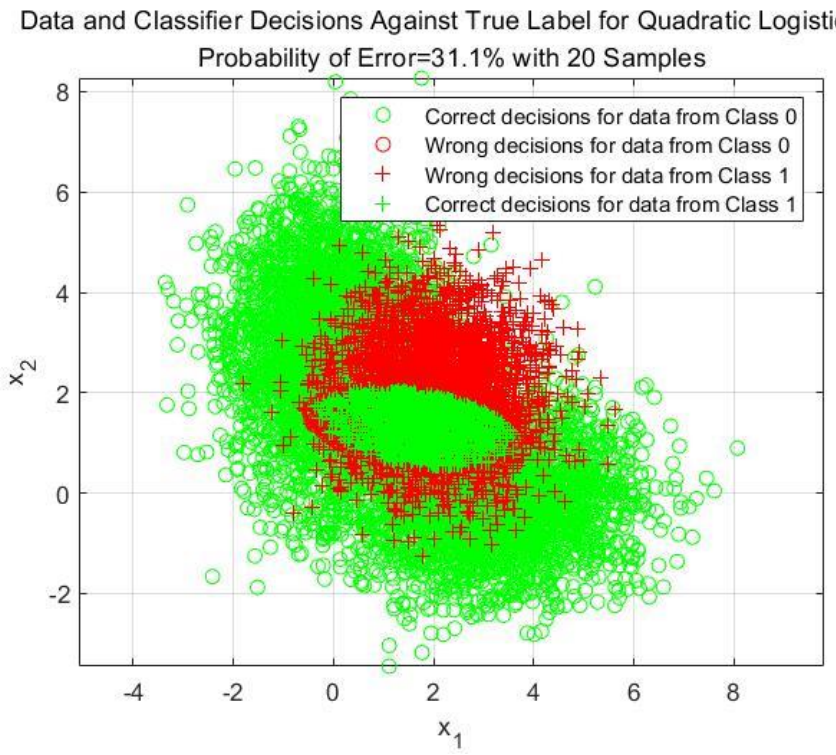
Samples	Linear
20	0.345
200	0.341
2000	0.341

(b) In this part maximum likelihood parameter estimation techniques were used to train 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 20, 200, and 2000 samples respectively. In the last, we will make a summary between these two classifier model to find how to make the probability of error minimized.

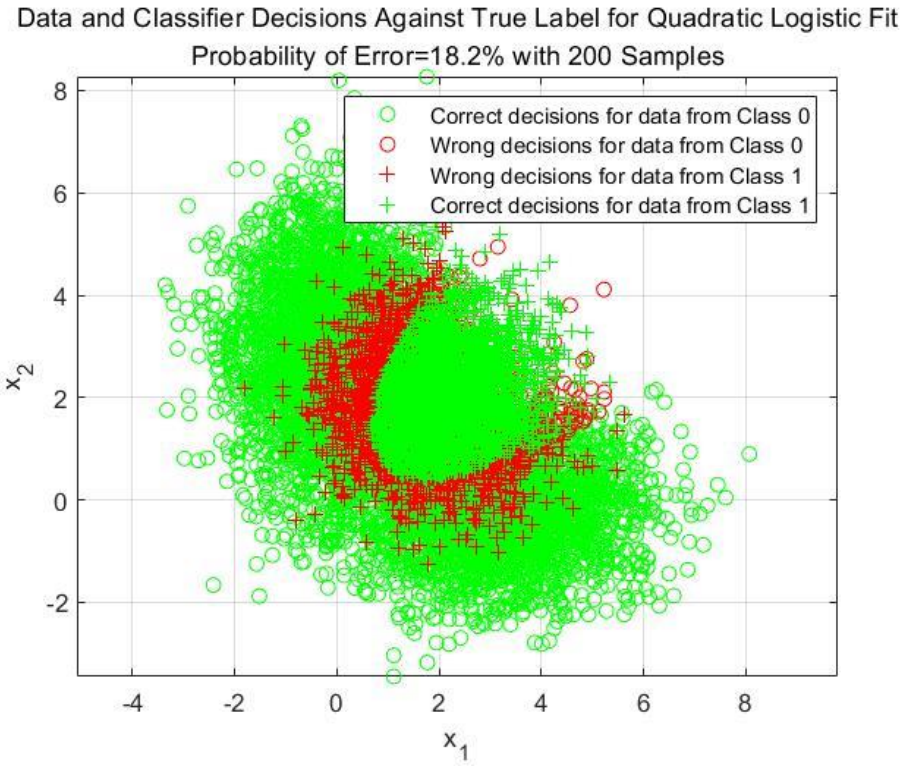
For the quadratic logistic function:

$$z(x) = [1 \;\; x_1 \;\; x_2 \;\; x_1^2 \;\; x_1x_2 \;\; x_2^x]^T$$

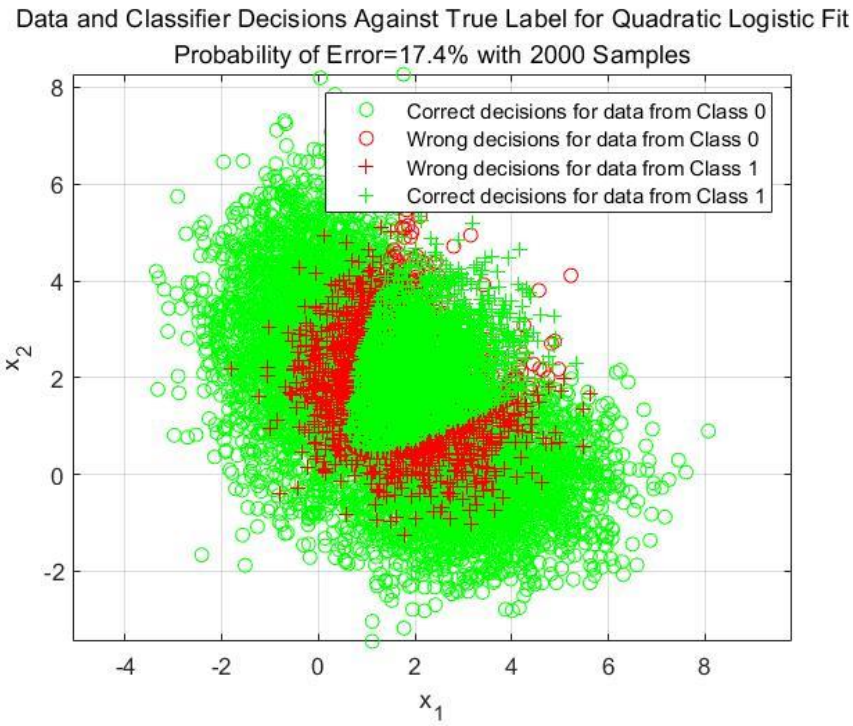
The next plot shows whether classifier decisions (Quadratic function) are correct or not using 20 samples:



The next plot shows whether classifier decisions (Quadratic function) are correct or not using 200 samples:



The next plot shows whether classifier decisions (Quadratic function) are correct or not using 2000 samples:



The table below 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 0.1661 obtained in Part 1.

Samples	Linear	Quadratic
20	0.345	0.311
200	0.341	0.182
2000	0.341	0.174

We can find that using a logistic-quadratic-function-based approximation of class label posterior functions can make the probability of error much lower than using logistic-linear-function-based approximations of class label posterior functions. Furthermore, if the size of samples is too small, like 20, might cause the result of classifier not stable.

Question2(40%)

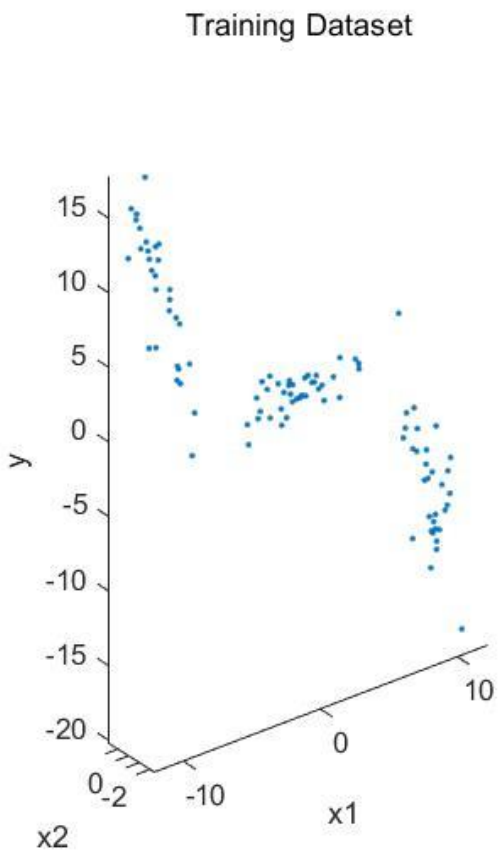
Assume that scalar-real y and two-dimensional real vector x are related to each other according to y = c(x,w)+v, where c(.,w) is a cubic polynomial in x with coefficients w and v is a random Gaussian random scalar with mean zero and σ^2-variance.

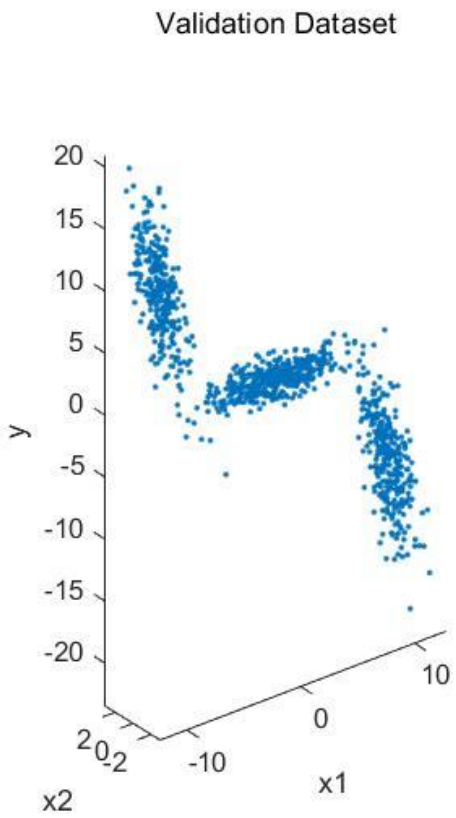
Given a dataset D = (x_1,y_1), . . . , (x_N,y_N) with N samples of (x,y) pairs, with the assumption

that these samples are independent and identically distributed according to the model, derive two estimators for w using maximum-likelihood (ML) and maximum-a-posteriori (MAP) parameter estimation approaches as a function of these data samples. For the MAP estimator, assume that w has a zero-mean Gaussian prior with covariance matrix γI.

Having derived the estimator expressions, implement them in code and apply to the dataset generated by the attached Matlab script. Using the training dataset, obtain the ML estimator and the MAP estimator for a variety of γ values ranging from 10^{-4} to 10^{4}. Evaluate each trained model by calculating the average-squared error between the y values in the validation samples and model estimates of these using c(.,$w_{trained}$). How does your MAP-trained model perform on the validation set as γ is varied? How is the MAP estimate related to the ML estimate? Describe your experiments, visualize and quantify your analyses (e.g. average squared error on validation dataset as a function of hyperparameter γ) with data from these experiments.

Answer:
Thanks to codes professor Deniz provided, I can get the train dataset (Ntrain=100) and validation dataset (Nvalidation=1000) like below:





First, we need to make sure how to find MAP estimate.
Let

$$\theta = \begin{bmatrix} x \\ y \end{bmatrix}$$

The pdf of θ (Gaussian prior) is:

$$p\begin{bmatrix} x \\ y \end{bmatrix} = 2\pi\sigma_x\sigma_y{}^{-1} e^{-\frac{1}{2}\begin{bmatrix} x \\ y \end{bmatrix}^T \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix}}$$

We need to find:

$$\theta_{map} = \begin{bmatrix} x_{map} \\ y_{map} \end{bmatrix} = argmax(\theta \mid R)$$

Where

$$R_i = \sqrt{(x_{Ti} - x_i)^2 + (y_{Ti} - y_i)^2}$$

(x_i, y_i) is the true data.
We use average squared error to do analysis of model performance. We do not consider the effect of noise.

$$\theta_{map} = argmax\ P(\theta|R)$$

Use Bayes Rule:

$$\theta_{map} = argmax\ \frac{P(R|\theta)P(\theta)}{P(R)}$$

$$= argmax\ P(R|\theta)P(\theta)$$

Use log function to

$$\theta_{map} = argmax\ [\ln P(R|\theta) + \ln P(\theta)]$$

$$= argmax\ [\ln \prod_{i=1}^{N} P(R_i|\theta) + \ln P(\theta)]$$

$$= argmax \sum_{i=1}^{N} \ln P(R_i|\theta) + \ln P(\theta)]$$

$$= argmax \sum_{i=1}^{N} \ln P(R_i|\theta) +$$

$$\ln((2\pi\sigma_x\sigma_y)^{-1} e^{-\frac{1}{2}\begin{bmatrix} x \\ y \end{bmatrix}^T \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}})]$$

Because we use Gaussian prior, so bring in the pdf of Gaussian distribution($R \sim N(0, \sigma^2)$):

$$\theta_{map} = argmax \sum_{i=1}^{N} \ln(\sqrt{2\pi\sigma_i^2})^{-1} e^{-\frac{(R_i)^2}{2\sigma_i^2}} +$$

$$\ln(2\pi\sigma_x\sigma_y)^{-1} - \frac{1}{2}\begin{bmatrix} x & y \end{bmatrix}\begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix}]$$

where $(\sqrt{2\pi\sigma_i^2})^{-1}$ is independent of θ

$$\theta_{map} = argmax \sum_{i=1}^{N} \ln\left(\sqrt{2\pi\sigma_i^2}\right)^{-1} -$$

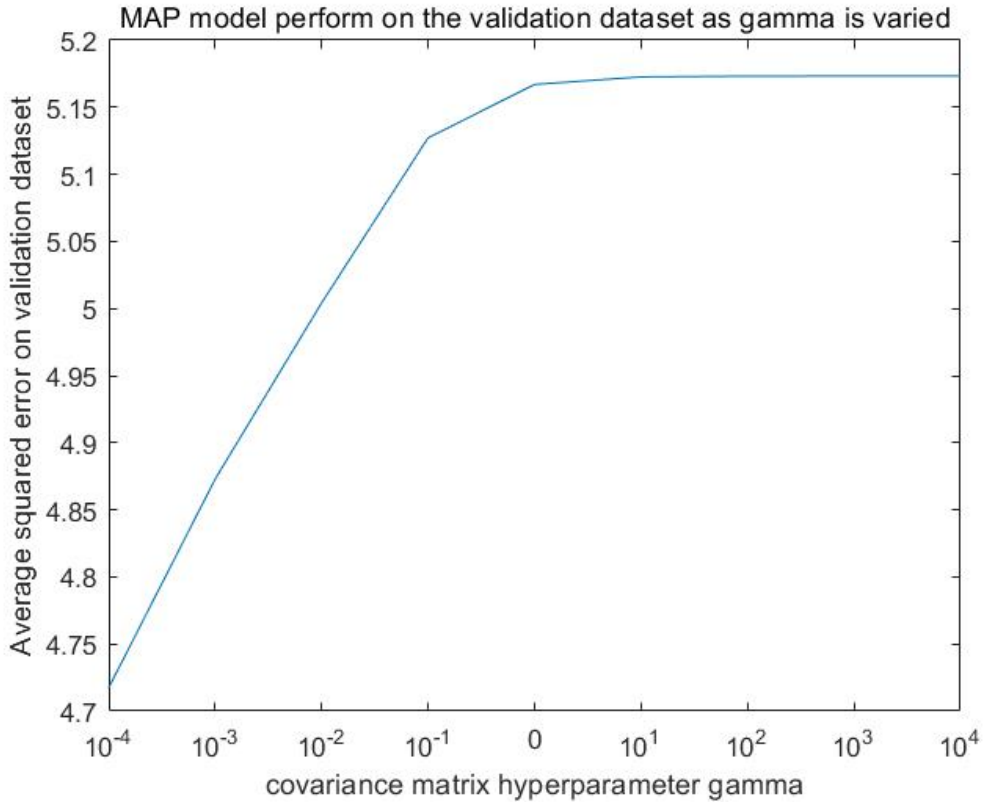
$$\frac{(\sqrt{(x_{Ti} - x_i)^2 + (y_{Ti} - y_i)^2})^2}{2\sigma_i^2} - \frac{1}{2}\begin{bmatrix} x & y \end{bmatrix}\begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix}]$$

where $(\sqrt{2\pi\sigma_i^2})^{-1}$ is independent of θ

$$\theta_{map} = argmin \sum_{i=1}^{N} \frac{(x_{Ti} - x_i)^2 + (y_{Ti} - y_i)^2}{\sigma_i^2} + (\frac{x^2}{\sigma_x^2} + \frac{y^2}{\sigma_y^2})$$

is the MAP estimate.

Using these algorithms (note that w has a zero-mean Gaussian prior with covariance matrix γ I), I can get a relationship between the hyperparameter γ and the average squared error:



From the plot, we can find that with γ increases, the performance of MAP model becomes worse because the average squared error increases, so we can get a conclusion that if the prior distribution of covariance is large, the performance of MAP model will be bad.

For the ML estimate:

$$\theta_{ml} = argmax\ P(R|\theta)$$

Use log function, we can get that:

$$\theta_{ml} = argmax\ \ln P(R|\theta)$$

$$= argmax \sum_{i=1}^{N} \ln P(R_i|\theta)$$

Bring in Gaussian distribution,

$$\theta_{ml} = argmax \sum_{i=1}^{N} \ln(\sqrt{2\pi\sigma_i^2})^{-1} e^{-\frac{(R_i)^2}{2\sigma_i^2}}$$

$$= argmax \sum_{i=1}^{N} \ln\left(\sqrt{2\pi\sigma_i^2}\right)^{-1} - \frac{(\sqrt{(x_{Ti}-x_i)^2+(y_{Ti}-y_i)^2})^2}{2\sigma_i^2}$$

where $(\sqrt{2\pi\sigma_i^2})^{-1}$ is independent of θ

So

$\theta_{ml} = argmin \sum_{i=1}^{N} \frac{(x_{Ti}-x_i)^2+(y_{Ti}-y_i)^2}{\sigma_i^2}$ is the ml estimate.

Running the code using the dataset we generate, we can get that

$\theta_{ml} = 4.982564$

Compared with the MAP estimate, we can find that the value of ML estimate is within the range of results. This can prove that ML classifier is a special case of MAP estimate. Comparing both MLE and MAP equation, the only thing differs is the inclusion of prior $P(\theta)$in MAP, otherwise they are identical.

Question3(20%)

Let Z be drawn from a categorical distribution (takes discrete values) with K possible outcomes/states and parameter θ, represented by $Cat(\Theta)$. Describe the value/state using a 1-of-K scheme for $\mathbf{z} = [z_1, \ldots, z_K]^T$ where $z_k = 1$ if variable is in state k and $z_k = 0$ otherwise. Let the parameter vector for the pdf be $\Theta = [\theta_1, \ldots, \theta_K]^T$, where $P(z_k = 1) = \theta_k$, for $k \in \{1, \ldots, K\}$.

Given $D\{\mathbf{z}_1, \ldots, \mathbf{z}_N\}$ with iid samples $\mathbf{z}_n \sim Cat(\Theta)$ for $n \in \{1, \ldots, N\}$:

- What is the ML estimator for Θ?
- Assuming that the prior $p(\Theta)$ for the parameters is a Dirichlet distribution with hyperparameter α, what is the MAP estimator for Θ?

Hint: The Dirichlet distribution with parameter α is

$$p(\Theta|\alpha) = \frac{1}{B(\alpha)} \prod_{k=1}^{K} \theta_k^{\alpha_k - 1} \quad \text{where the normalization constant is} \quad B(\alpha) = \frac{\prod_{k=1}^{K} \Gamma(\alpha_k)}{\Gamma(\sum_{k=1}^{K} \alpha_k)}$$

Answer:

$$\widehat{\Theta_{ML}} = argmax \ \ln P(D|\theta)$$

$$= argmin -\frac{1}{N} \sum_{i=1}^{N} \ln P(z_i|\theta)$$

$$= argmin -\frac{1}{N}\sum_{i=1}^{N} \ln \theta_{z_i}$$

Assume $\theta \geq 0$ will be inactive.

$s.t.\ \ \theta^{\tau}\mathrm{I} - 1 = 0$

$$\varsigma(\theta, \lambda) = -\frac{1}{N}\sum_{i=1}^{N} \ln \theta_{z_i} - \lambda(\theta^{\tau}\mathrm{I} - 1)$$

$$0 = \frac{d}{d\theta_l} = -\frac{1}{N}\sum_{i=1}^{N} \frac{d}{d\theta_l} \ln \theta_{z_i} - \lambda \frac{d}{d\theta_l}(\theta^{\tau} I)$$

$$\delta_{z_l} = \begin{cases} 0 & if\ z \neq l \\ 1 & if\ z = l \end{cases}$$

$$0 = \frac{1}{N}\sum_{i=1}^{N} \frac{d}{d\theta_l} \delta_{z_i k} - \lambda$$

$$= \frac{N_l}{N\theta_l} - \lambda$$

$$\lambda(\theta_1 + \cdots + \theta_k) = \frac{N_1 + \cdots N_k}{N}$$

$$\lambda = 1$$

So $\theta_l = \frac{N_i}{N} = \bar{N}$ is the maximum likelihood estimate.

$$\widehat{\Theta_{MAP}} = argmax\ \ln P(\theta|D)$$

$$\widehat{\Theta_{MAP}} = argmax\ \frac{1}{N}\sum_{i=1}^{N} \ln \theta_{z_i} + \ln \frac{\prod_{l=1}^{K} \theta_l^{\alpha_l - 1}}{B(\alpha)}$$

$$\varsigma(\theta, \lambda) = -\frac{1}{N}\sum_{i=1}^{N} \ln \theta_{z_i} + \sum_{l=1}^{K} (\alpha_l - 1) \ln \theta_l - \lambda(\theta^{\tau}\mathrm{I} - 1)$$

Use $\frac{d}{d\theta_l}$ to calculate the argmax:

$$0 = \frac{N_l}{N\theta_l} + \sum_{l=1}^{K} \frac{(\alpha l - 1)}{\theta_l} - \lambda$$

$$\lambda(\theta_1 + \cdots + \theta_k) = 1 + \alpha^{\tau} I - k$$

$$\lambda = (\alpha^{\tau} + 1 - k)$$

So $\theta_l = \frac{\left(\frac{N_l}{N}\right) + (\alpha^{\tau} I - k)}{(\alpha^{\tau} I - k + 1)}$ is the maximum-a-posteriori estimate.

Codes

Question1(Matlab)

```
% Code help and example from Prof.Deniz

clear all;close all;clc;

%%=========================Setup=========================%%
dimension=2; %Dimension of data
%Define data
D.d20.N=20;
D.d200.N=200;
D.d2k.N=2000;
D.d10k.N=10000;
dTypes=fieldnames(D);
%Define Statistics
p=[0.65 0.35]; %Prior
%Label 0 GMM Stats
mu0=[3 0;0 3]';
Sigma0(:,:,1)=[2 0;0 1];
Sigma0(:,:,2)=[1 0;0 2];
alpha0=[0.5 0.5];
%Label 1 Single Gaussian Stats
mu1=[2 2]';
Sigma1=[1 0;0 1];
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.d10k.x,alpha0,mu0,Sigma0);
px1=evalGaussian(D.d10k.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.d10k.labels,D.d10k.N0,D.d10k.N1,D.d10k.phat);
logGamma_ideal=log(p(1)/p(2));
decision_ideal=discScore>logGamma_ideal;
p10_ideal=sum(decision_ideal==1 & D.d10k.labels==0)/D.d10k.N0;
p11_ideal=sum(decision_ideal==1 & D.d10k.labels==1)/D.d10k.N1;
pFE_ideal=(p10_ideal*D.d10k.N0+(1-p11_ideal)*D.d10k.N1)/(D.d10k.N0+D.d10k.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.d10k.x,D.d10k.labels,decision_ideal);
plotERMContours(D.d10k.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(4,10001,3);
samples=[20 200 2000 10000];
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.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.d10k.labels,...
        D.d10k.N0,D.d10k.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
    %plotMinPFE(logGamma,prob.pFE,prob.min_pFE_ind);
    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;

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.d10k.N); D.d10k.x];
    gamma=0;
```

```
    lin.prob=CalcProb(lin.discScore,gamma,D.d10k.labels,...
        D.d10k.N0,D.d10k.N1,D.d10k.phat);
    % quad.decision=[ones(D.d10k.N,1) D.d10k.x]*quad.theta>0;
    plotDecisions(D.d10k.x,D.d10k.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))));
    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.d10k.N); D.d10k.x; D.d10k.x(1,:).^2;...
        D.d10k.x(1,:).*D.d10k.x(2,:); D.d10k.x(2,:).^2];
    quad.discScore=quad.theta'*quad.xScore;
    gamma=0;
    quad.prob=CalcProb(quad.discScore,gamma,D.d10k.labels,...
        D.d10k.N0,D.d10k.N1,D.d10k.phat);
    plotDecisions(D.d10k.x,D.d10k.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
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Question2(Python)

```
import numpy as np
import matplotlib.pyplot as plt


def hw2q1():
    Ntrain = 100
    data = generateData(Ntrain)
    plot3(data[0,:],data[1,:],data[2,:],'Training Dataset')
    xTrain = data[0:2,:]
    yTrain = data[2,:]

    Ntrain = 1000
    data = generateData(Ntrain)
    plot3(data[0,:],data[1,:],data[2,:],'Validation Dataset')
    xValidate = data[0:2,:]
    yValidate = data[2,:]

    return xTrain,yTrain,xValidate,yValidate

def generateData(N):
    gmmParameters = {}
    gmmParameters['priors'] = [.3,.4,.3] # priors should be a row vector
    gmmParameters['meanVectors'] = np.array([[-10, 0, 10], [0, 0, 0], [10, 0, -10]])
    gmmParameters['covMatrices'] = np.zeros((3, 3, 3))
    gmmParameters['covMatrices'][:,:,0] = np.array([[1, 0, -3], [0, 1, 0], [-3, 0, 15]])
    gmmParameters['covMatrices'][:,:,1] = np.array([[8, 0, 0], [0, .5, 0], [0, 0, .5]])
    gmmParameters['covMatrices'][:,:,2] = np.array([[1, 0, -3], [0, 1, 0], [-3, 0, 15]])
    x,labels = generateDataFromGMM(N,gmmParameters)
    return x

def generateDataFromGMM(N,gmmParameters):
#     Generates N vector samples from the specified mixture of Gaussians
#     Returns samples and their component labels
#     Data dimensionality is determined by the size of mu/Sigma parameters
    priors = gmmParameters['priors'] # priors should be a row vector
    meanVectors = gmmParameters['meanVectors']
    covMatrices = gmmParameters['covMatrices']
    n = meanVectors.shape[0] # Data dimensionality
    C = len(priors) # Number of components
    x = np.zeros((n,N))
    labels = np.zeros((1,N))
```

```
        # Decide randomly which samples will come from each component
        u = np.random.random((1,N))
        thresholds = np.zeros((1,C+1))
        thresholds[:,0:C] = np.cumsum(priors)
        thresholds[:,C] = 1
        for l in range(C):
                indl = np.where(u <= float(thresholds[:,l]))
                Nl = len(indl[1])
                labels[indl] = (l+1)*1
                u[indl] = 1.1
                x[:,indl[1]]      =      np.transpose(np.random.multivariate_normal(meanVectors[:,l],
covMatrices[:,:,l], Nl))

        return x,labels

def plot3(a,b,c,title,mark="o",col="b"):
    from matplotlib import pyplot
    import pylab
    from mpl_toolkits.mplot3d import Axes3D
    pylab.ion()
    fig = pylab.figure()
    ax = Axes3D(fig)
    ax.scatter(b, a, c,marker=mark,color=col)
    ax.set_xlabel("x2")
    ax.set_ylabel("x1")
    ax.set_zlabel("y")
#    ax.set_aspect('equal')
    set_aspect_equal_3d(ax)
    ax.set_title(title)


def set_aspect_equal_3d(ax):
        """Fix equal aspect bug for 3D plots."""

        xlim = ax.get_xlim3d()
        ylim = ax.get_ylim3d()
        zlim = ax.get_zlim3d()

        from numpy import mean
        xmean = mean(xlim)
        ymean = mean(ylim)
        zmean = mean(zlim)

        plot_radius = max([abs(lim - mean_)
```

```
                    for lims, mean_ in ((xlim, xmean),
                                        (ylim, ymean),
                                        (zlim, zmean))
                    for lim in lims])

    ax.set_xlim3d([xmean - plot_radius, xmean + plot_radius])
    ax.set_ylim3d([ymean - plot_radius, ymean + plot_radius])
    ax.set_zlim3d([zmean - plot_radius, zmean + plot_radius])

def little_phi(x):
    return np.array([1, x[0], x[1], x[0]**2, x[1]**2, x[0]*x[1], x[0]**3, x[1]**3, x[0]**2*x[1], x[1]**2*x[0]])

def big_phi(data):
    return np.transpose(np.apply_along_axis(little_phi, 0, data))

def w_hat_ml(x, y):
    bigphi = big_phi(x)
    bigphi_transpose = np.transpose(bigphi)
    return np.linalg.inv(bigphi_transpose @ bigphi) @ bigphi_transpose @ y

def w_hat_map(x, y, gamma):
    bigphi = big_phi(x)
    bigphi_transpose = np.transpose(bigphi)
    return  np.linalg.inv(1/gamma*np.eye(bigphi.shape[1])+bigphi_transpose @ bigphi) @ bigphi_transpose @ y

def w_hat_map_manygammas(x,y,gammas):
    result = np.empty((10, gammas.shape[0]))
    for i in range(gammas.shape[0]):
        result[:,i] = w_hat_map(x,y, gammas[i])
    return result

def c(x,w):
    return np.transpose(w) @ little_phi(x)

def mse(y1, y2):
    return ((y1-y2)**2).mean(axis=0)


# xtrain, ytrain, xvalidate, yvalidate = hw2q1()
xtrain = np.load(r'C:\Users\eckmb\OneDrive - Northeastern University\Courses\EECE5564\HW2\p1_xtrain.npy')
ytrain = np.load(r'C:\Users\eckmb\OneDrive - Northeastern
```

```
University\Courses\EECE5564\HW2\p1_ytrain.npy')
xvalidate = np.load(r'C:\Users\eckmb\OneDrive - Northeastern University\Courses\EECE5564\HW2\p1_xvalidate.npy')
yvalidate = np.load(r'C:\Users\eckmb\OneDrive - Northeastern University\Courses\EECE5564\HW2\p1_yvalidate.npy')


w_hat_ml_train = w_hat_ml(xtrain, ytrain)
gammas = np.power(np.array([10.0]), np.arange(-4, 5))
w_hat_map_train = w_hat_map_manygammas(xtrain, ytrain, gammas)

ml_estimatedy = np.apply_along_axis(c, 0, xvalidate, w=w_hat_ml_train)
map_estimatedy = np.transpose(np.apply_along_axis(c, 0, xvalidate, w=w_hat_map_train))

ml_mse = mse(yvalidate, ml_estimatedy)
map_mse = mse(np.transpose(np.tile(yvalidate,[gammas.shape[0], 1])), map_estimatedy)

plt.plot(gammas, map_mse, 'ro')
plt.title("MAP model perform on the validation dataset as gamma is varied")
plt.xlabel("convariance matrix hyperparameter gamma")
plt.ylabel("Average squared error on validation dataset")
```