Assignment 2
Juan Pablo Bernal
5/28/2020

1.

For the first question, 4 data sets were generated from 2 classes(figure.1); a Gaussian mixture model (with a class prior of 0.65 and class label 1), and a single Gaussian (with a class prior of 0.35 and class label of 2). The 4 data sets contained respectively 50, 500, 5k, and 10k samples. The first 3 data sets were used to train the algorithms and the 10k data set was used to validate the estimated models. For each data set, the true class label of each sample and the #of samples that came from each class were recorded.

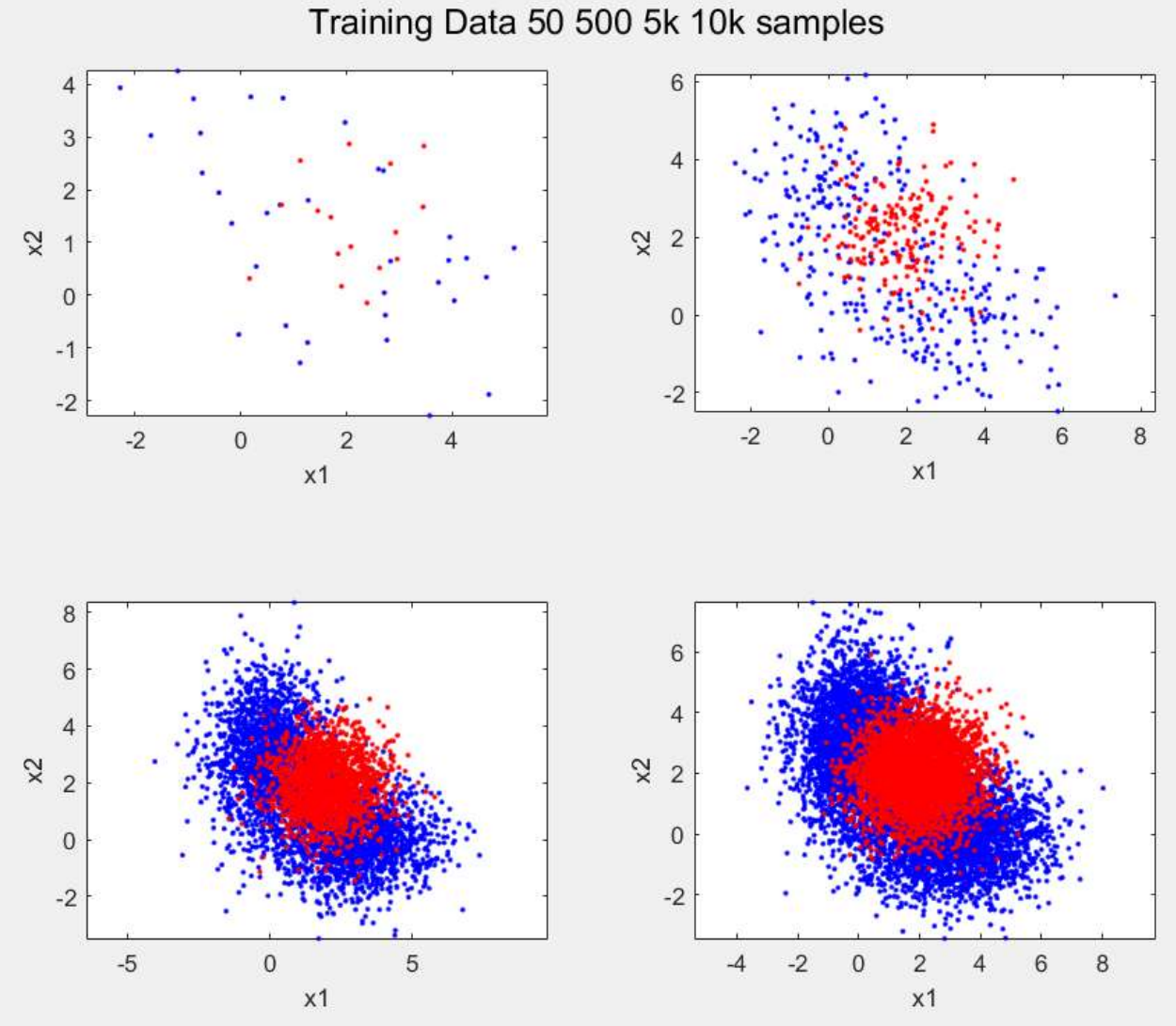

Figure. 1 Data Sets

Part 1.

Theoretically, the classifier that achieves the minimum probability of error would be ERM with 0-1 loss, which would reduce to a MAP classifier. The MAP classification rule is $D = argmax_l\ ln(P(x|L = l) * P(l))$ which decides that a sample came from the class that results in the maximum likelihood keeping in mind the respective class priors (validate function in the code). To find the min-P error achievable with the generated data, the true pdfs used to generate the data were applied to the 10k data set. Then the probabilities of the 4 possible outcomes (TP, FP, TN, FN; positive class2, negative class1) were calculated by counting the decisions made vs the true labels with $\frac{\#samples\ of\ (D = i|L = j)}{N_j}$ where N_j is the true number of samples generated from class j, D is the decision made and L is the true label. The min-P error was computed using the equation $P(FP) * P(L = 1) + P(FN) * P(L = 2)$ and found to be 0.179 or 17.9%. Then to generate the ROC curve, since there are only 2 classes, the decision rule was modified to be able to compare it to a threshold. The new rule was if $ln(\frac{P(x|L=1}{P(x|L=2}) \leq \gamma$ decide class 2 and if $ln(\frac{P(x|L=1}{P(x|L=2}) > \gamma$ decide class 1. Gamma was gradually increased and for each, the probabilities of FP, TP, and FN were computed to plot the ROC curve (figure.3) and to calculate the probability of the min-P error using the same method for each threshold. The location of the gamma that gave the min-P error was marked in the ROC curve with a red +(a red circle was placed around it to make the + easier to find). It was found that the min-P error achievable was 0.178. Or 17.8%. In both cases, it was almost the same only with a difference od 0.001 (values from figure.2)

```
Part 1
Theoretical min-P error =0.17878
ROC min-P error =0.17817
```

Figure. 2 Output of Part 1 code

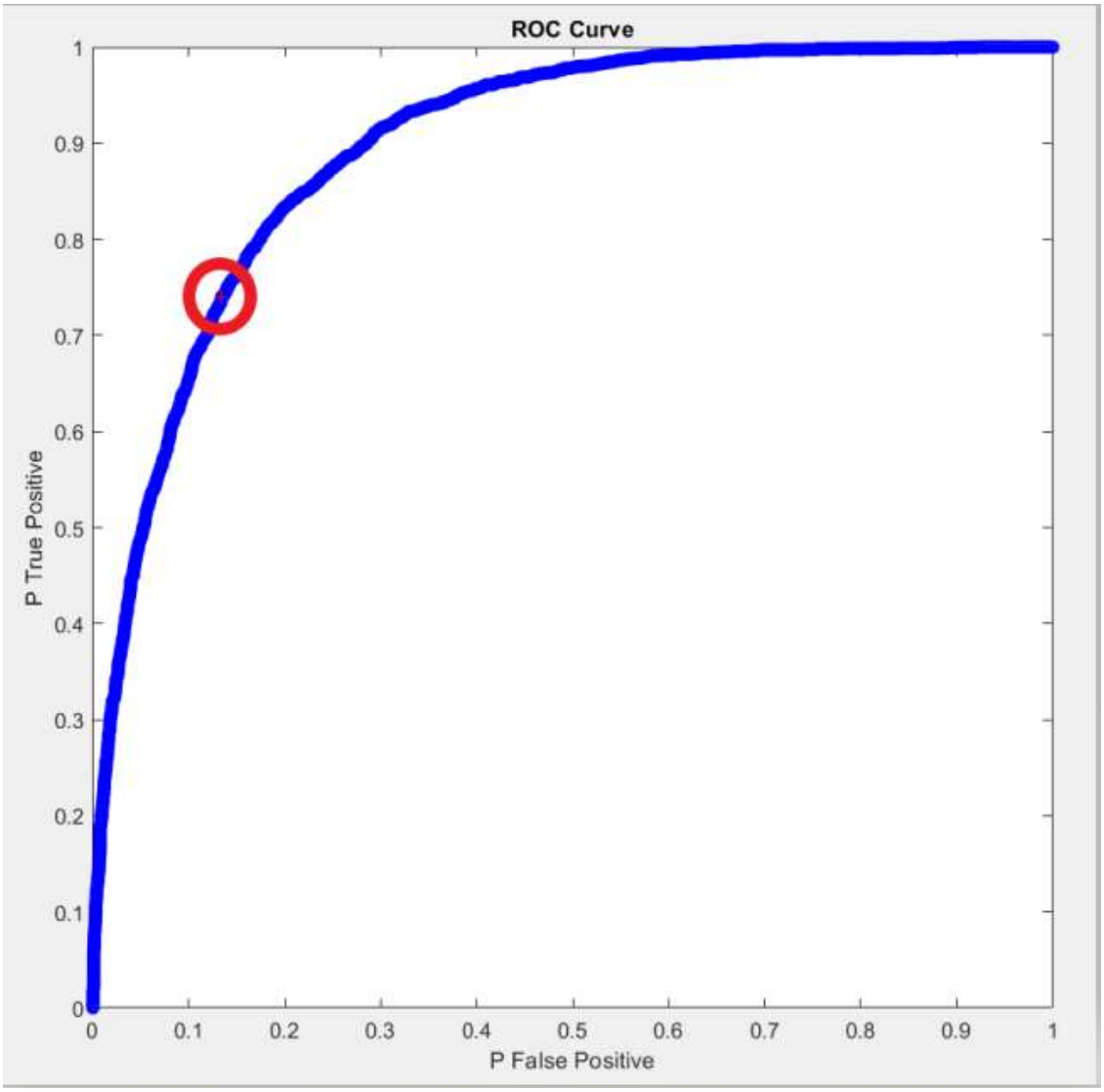

Figure 3. ROC curve Part 1

Part 2.

The same procedure from Part 1 was done; the validation using the 10k data set, generating the ROC curve, and finding the min-P error both theoretically and from the ROC curve. However, this time the parameters of the pdfs used were estimated instead of using the truth. The parameters to estimate were mu and covariance for the single Gaussian (class2) and alpha, mu, and covariance for the GMM (class1) knowing that there where 2 Gaussians in the mixture model. This estimation was done with the training data sets for which the label of the samples and the number of samples from each class were known. Moreover, the procedure was done three times, one with each training data set (50 500 and 5k samples) to test the effect that the number of samples has on the accuracy of the estimation.

For the single Gaussian, the mu was estimated by using the sample average (MLE derived in class) $\mu_{ML} = \frac{1}{N_2}\sum_{i=1}^{N2} x_i$ where N1 is the total number of samples that came from class 2 and xi are the samples that are known to be from class 2. In the code, this was done with the MatLab builtin function mean(). For the covariance, the sample covariance (MLE derived in class) was used $\Sigma_{ML} = \frac{1}{N_2}\sum_{i=1}^{N2}(x_i - \mu_{ML}) * (x_i * \mu_{ML})^T$ In the code this was done with the MatLab builtin function cov().

For the GMM, the estimation of the parameters was done using the ML algorithm from the Bilmes Tutorial (code was taken from the GMforGMM.m). In this algorithm, the parameters are estimated by recursively applying the ML estimators for alpha, mu, and covariance. The equations used are shown in the figure. 5 (Bilmes). The parameters are initialized as follows: Alpha as all the components having the same class prior, mu is by randomly taking M (number of components) samples and finding the mean of this set, and covariance by finding the covariance of the same random set of samples. In each iteration, α_l^{new} is calculated using the previously estimated values, then this result is used to calculate μ_l^{new} and finally, the result is used to calculate Σ_l^{new} . The algorithm is recursively run until the difference between the current value and the last is less than a threshold delta or is less than 10000 iteration to prevent an infinite loop. In the code, each training set has a different delta that was determined by try and error.

Results for each training data can be found in the figure. 4. Part B.a was calculated with the 5k samples data set, PartB.b with 500 samples data set, and PartB.c with 50 samples data set. Respectively, the ROC curves can be found in figures 6 to 8, The location of the gamma that gave the min-P error was marked in the ROC curve with a red + (a red circle was placed around it to make the + easier to find). Finally, by comparing the minimum probability of error given by each estimation, we can conclude that as the training set gets larger the accuracy of the estimation gets closer to the real pdfs. With 5k samples, we get a min-P error of 0.178 same as we got in Part 1, and 500 and 50 this probability increases to 0.182 and 0.195

```
part B.a
Theoretical min-P error =0.17967
ROC min-P error =0.17794
part B.b
Theoretical min-P error =0.18297
ROC min-P error =0.18199
part B.c
Theoretical min-P error =0.19613
ROC min-P error =0.19554
```

Figure. 4 Output of Part 2 code

$$\alpha_\ell^{new} = \frac{1}{N}\sum_{i=1}^{N} p(\ell|x_i, \Theta^g)$$

$$\mu_\ell^{new} = \frac{\sum_{i=1}^{N} x_i p(\ell|x_i, \Theta^g)}{\sum_{i=1}^{N} p(\ell|x_i, \Theta^g)}$$

$$\Sigma_\ell^{new} = \frac{\sum_{i=1}^{N} p(\ell|x_i, \Theta^g)(x_i - \mu_\ell^{new})(x_i - \mu_\ell^{new})^T}{\sum_{i=1}^{N} p(\ell|x_i, \Theta^g)}$$

Figure. 5 ML estimator for GMM parameters

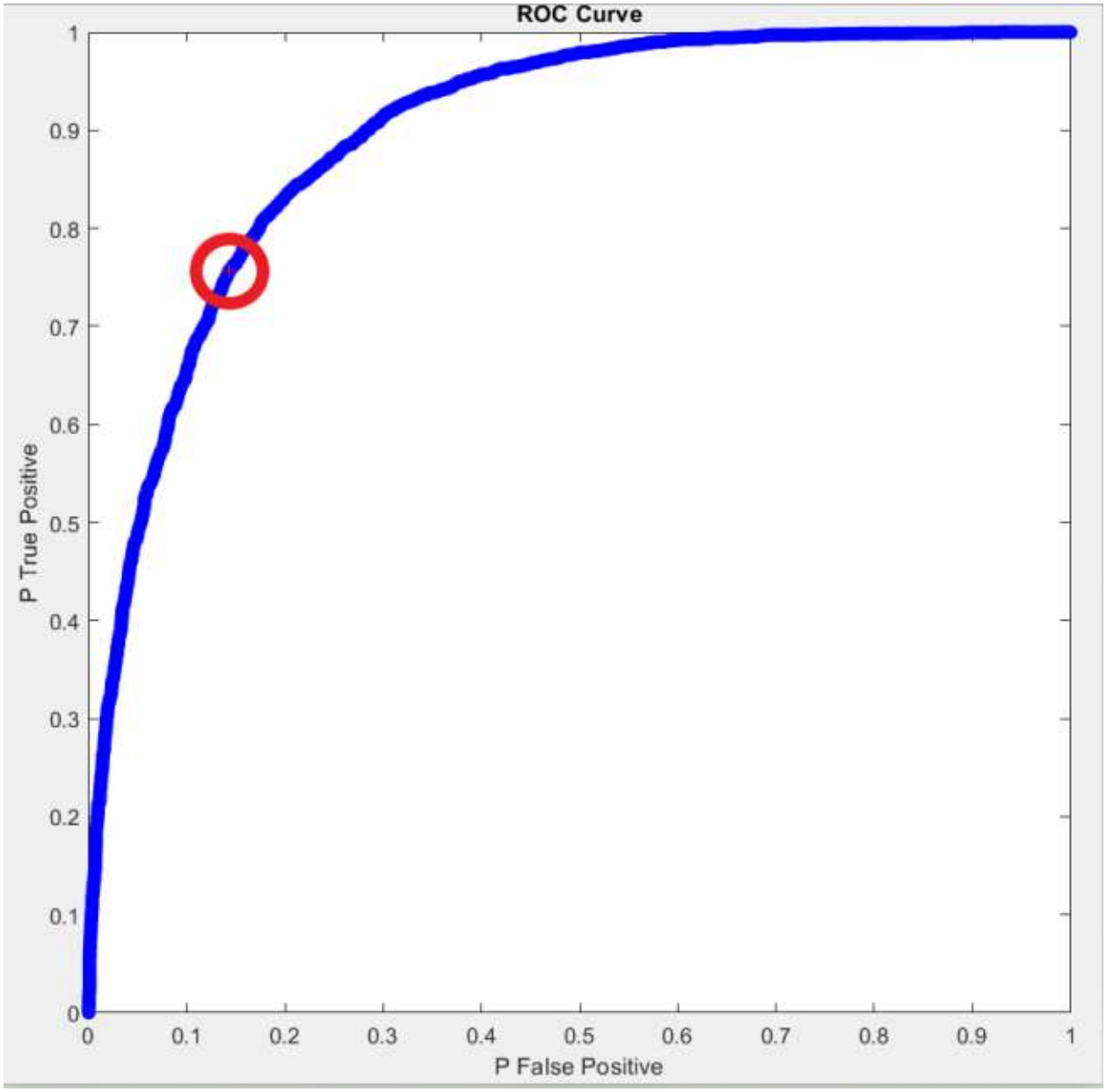

Figure. 6 ROC curve PartB.aTrained with 5k Samples

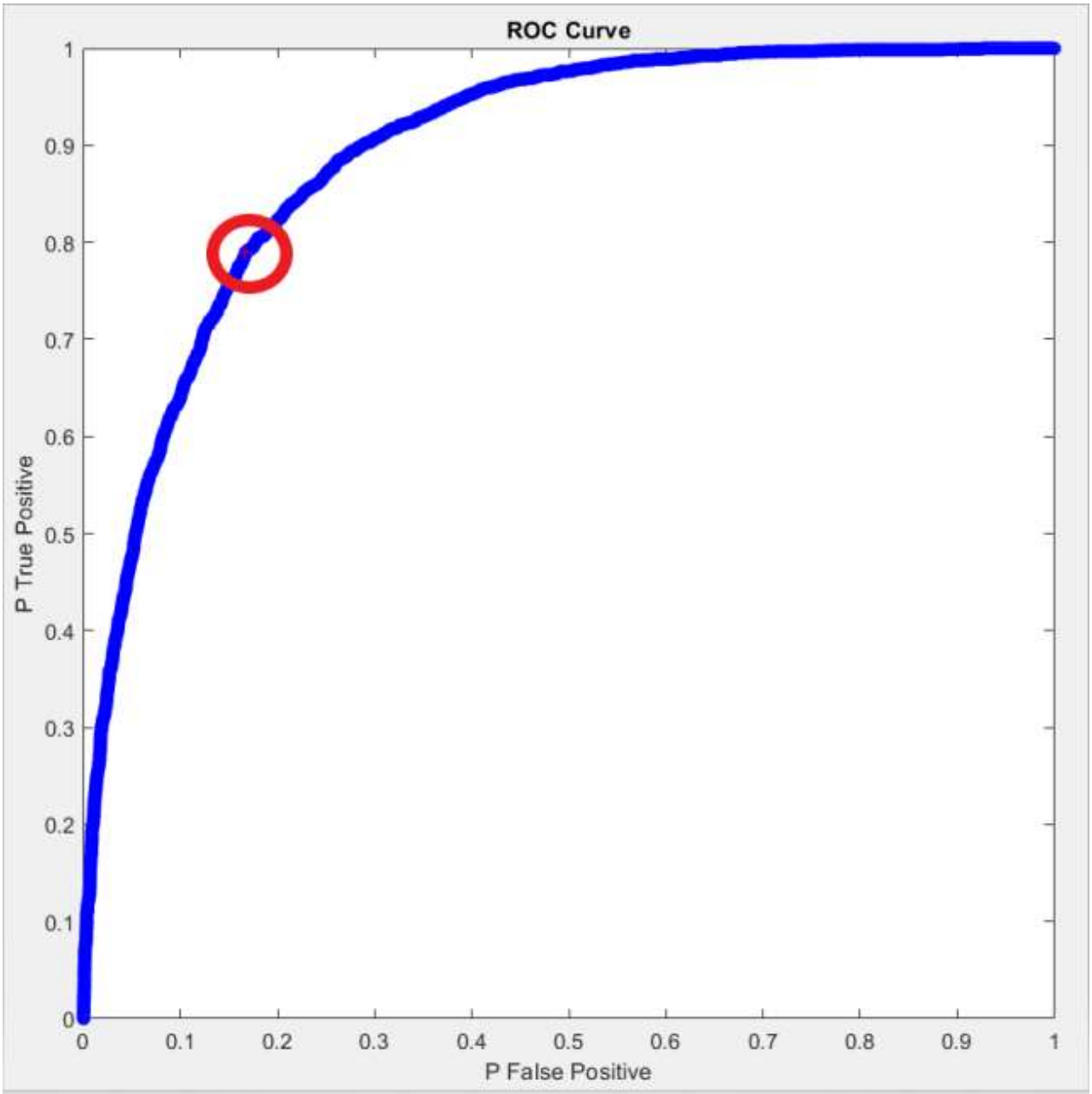

Figure. 7 ROC curve PartB.b Trained with 500 Samples

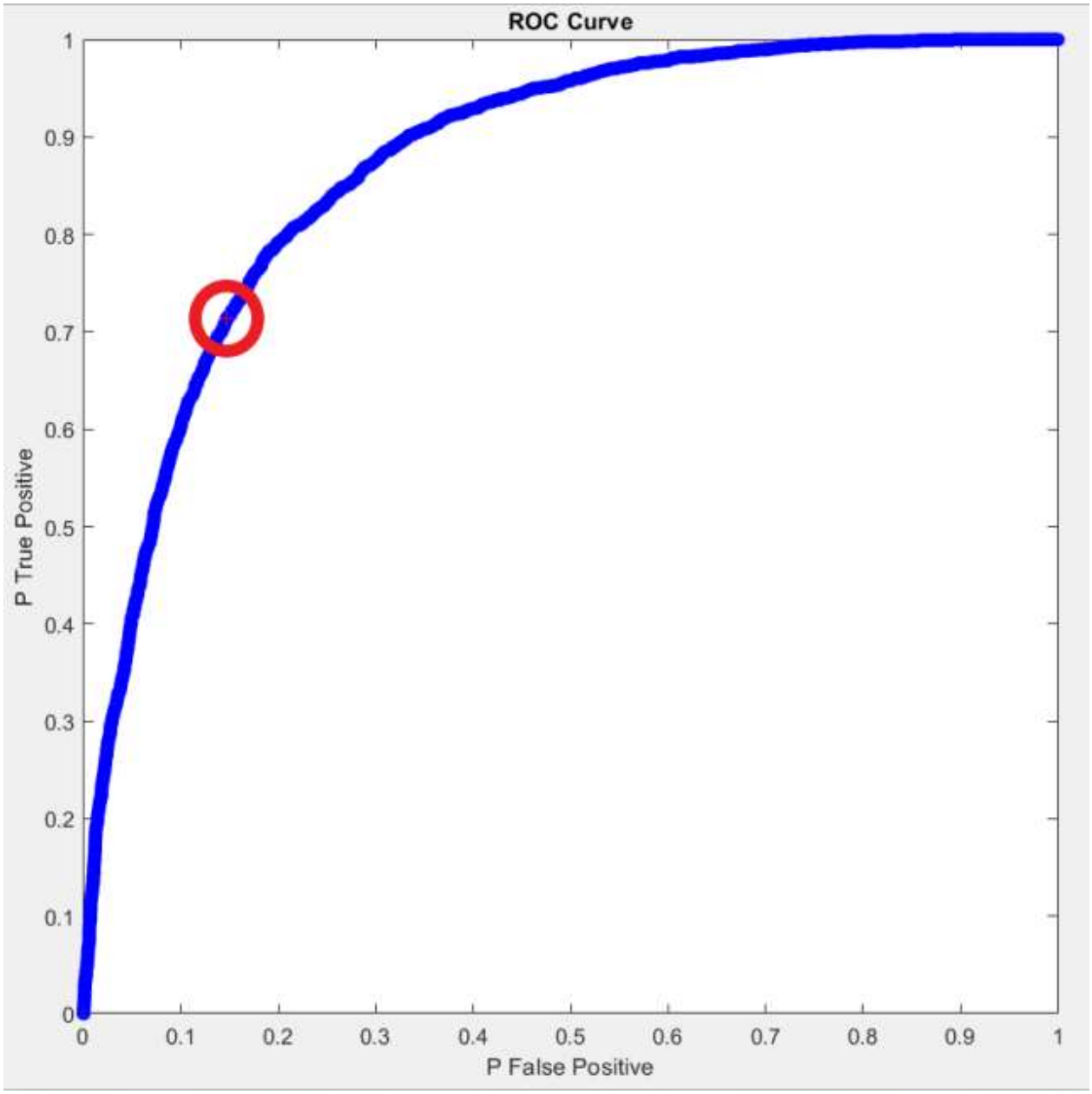

Figure. 8 ROC curve PartB.c Trained with 50 Samples

Part 3.

The MAP classification rule originally asks to maximize the class posterior, the rule used before is modified by using Bayes rule. Therefore, this time the class posteriors were estimated using a logistic-linear-function based approximation and logistic-quadratic-function based approximation. Now the decision rule becomes $D = argmax_l P(L = l \mid x)$. The approximation made for the class2 posterior was $H = \frac{1}{1+e^{-w^t b(x)}}$ where b(x) is a vector containing the data set and w is the

parameter to be optimized which has the same dimension as b(x); the class2 posterior is estimated with $1-H$. The ML estimation formula used to optimize the parameter w was: $W_{ML} = argmin_w -\frac{1}{N}\sum_{i=1}^{N}[(label-1)*ln(H)+(1-(label-1))*ln(1-H)]$ there is an extra -1 in the labels because the labels used when de data was generated are 1-2 and for this equation to work we need 0-1 labels. For the logistic-linear-function b(x) has the form $[1\ x_1\ x_2]^T$ and for the logistic-quadratic-function b(x) has te form $[1\ \ x_1\ \ x_2\ \ x_1{}^2\ \ x_1x_2\ \ x_2{}^2]^T$. The function was optimized using MatLab fminsearch() numerical optimization function. Finally, both estimation models were run with the 3 training data sets and validated with the 10k samples data set.

The results are shown in Figures 9 and 10 respectively for each approximation model. The linear model gave a probability of error of around 0.35 for all training data sets; meaning that this approximation can't do better than that. After all, it is a simplistic approximation and very restricted. On the other hand, with a more complex approximation, the quadratic, we see that the min-P error gets closer and closer to the theoretical minimum value we found in part 1. Also, as the number of samples increases the probability gets lower and closer to the theoretical. 0.246 with 50 samples, 0.186 with 500 samples, and 0.179 with 5k samples (same value as found in part 1).

```
50 samples
min-P error =0.35261
500 samples
min-P error =0.35407
5k samples
min-P error =0.35158
```

Figure. 9 output of Logistic-Linear-Function based approximations

```
50 samples
min-P error =0.2463
500 samples
min-P error =0.18578
5k samples
min-P error =0.1797
```

Figure.10 output of Logistic-quadratic-Function based approximations

2.

In question 2, three training data sets were generated from a Gaussian mixture model with 4 simple Gaussians. The sizes of the sets were 2k, 20k, and 200k samples (figure.11). 100 experiments were run and for each experiment, new datasets were generated. Figure. 11 only displays one of the sets to show how the data is distributed.

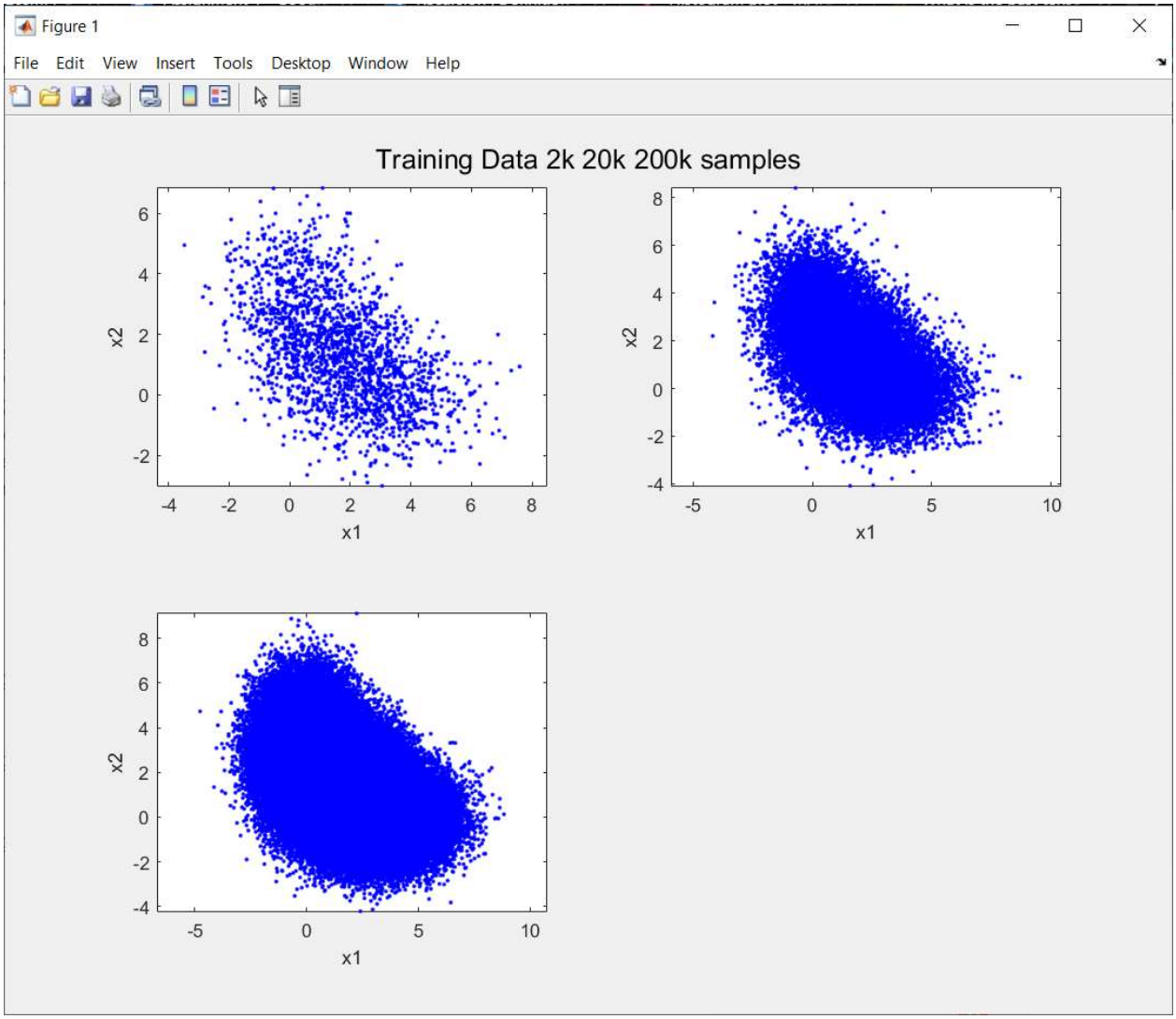

Figure. 11 One of the Datasets

This time, the idea was to estimate the order of the model (how many simple Gaussians there were) and see how the number of samples affects the estimate. To achieve this, the BIC method was used, this method proposes that the order of the GMM is the one that minimizes the equation $BICscore = -2\sum_{i=1}^{N} ln(P_m(x_i \mid \theta^n_{ML})) + n * ln(N)$ where P_m the pdf of the GMM with m components, n is the number of parameters (6m-1 was used as recommended in the assignment), N the number of samples (2*size(x) was used as recommended in the assignment), x the data set, and θ^n_{ML} the ML estimate of the

parameters which were estimated using the same ML algorithm designed in questin1-Par2. In each experiment, for each dataset(2k 20k and 200k), the BIC score was calculated for GMMs with 1 to 10 Gaussian components. When all the scores had been calculated, the one that gave the minimum score was selected and a counter for that selected model was incremented. After the 100 experiments were completed a histogram showing how many times a model order had been selected was generated for each dataset (figures 12 to 14) and the final decision was the model with more counts; in case there were multiple with the same number of votes (min or max) the model with the lower order was selected. Smaller complexity is always preferred. Finally, a new GMM was estimated (using the same ML algorithm) with the selected model order for each dataset and the equilevel contours were overlaid over its respective dataset (figure.15). Also the same was done for the model that got the least number of votes(figure.16).

As we can see from the histograms and the equilevel contours, as the size of the sample data increased the accuracy of the BIC increased. For 2k samples, we got that the optimal model order was 2 with 72 votes. Then with 20k samples, we got the correct answer that it should be of order 4 with 22 votes. Lastly, with 200k samples, we got that the order should be 8, however, order 4 got a lot of votes and by looking at the equilevel contours in figure 15 we can see that with 8 is also a very good fit for the data generated. To contrast the order that got more votes to the one that got the least, in figure 15 and 16 we can see that in figure 15 the equilevel contours are fit very accurately the data generated. On the other hand, in figure 16 we can see that the data falls outside the contours and does not follow them closely for 20k and 200k and is too complex for the 2k data. The order of the models that got the minimum votes in figure 16 are: 1 for 200k samples, 1 for 20k samples, and 5 for 2k samples.

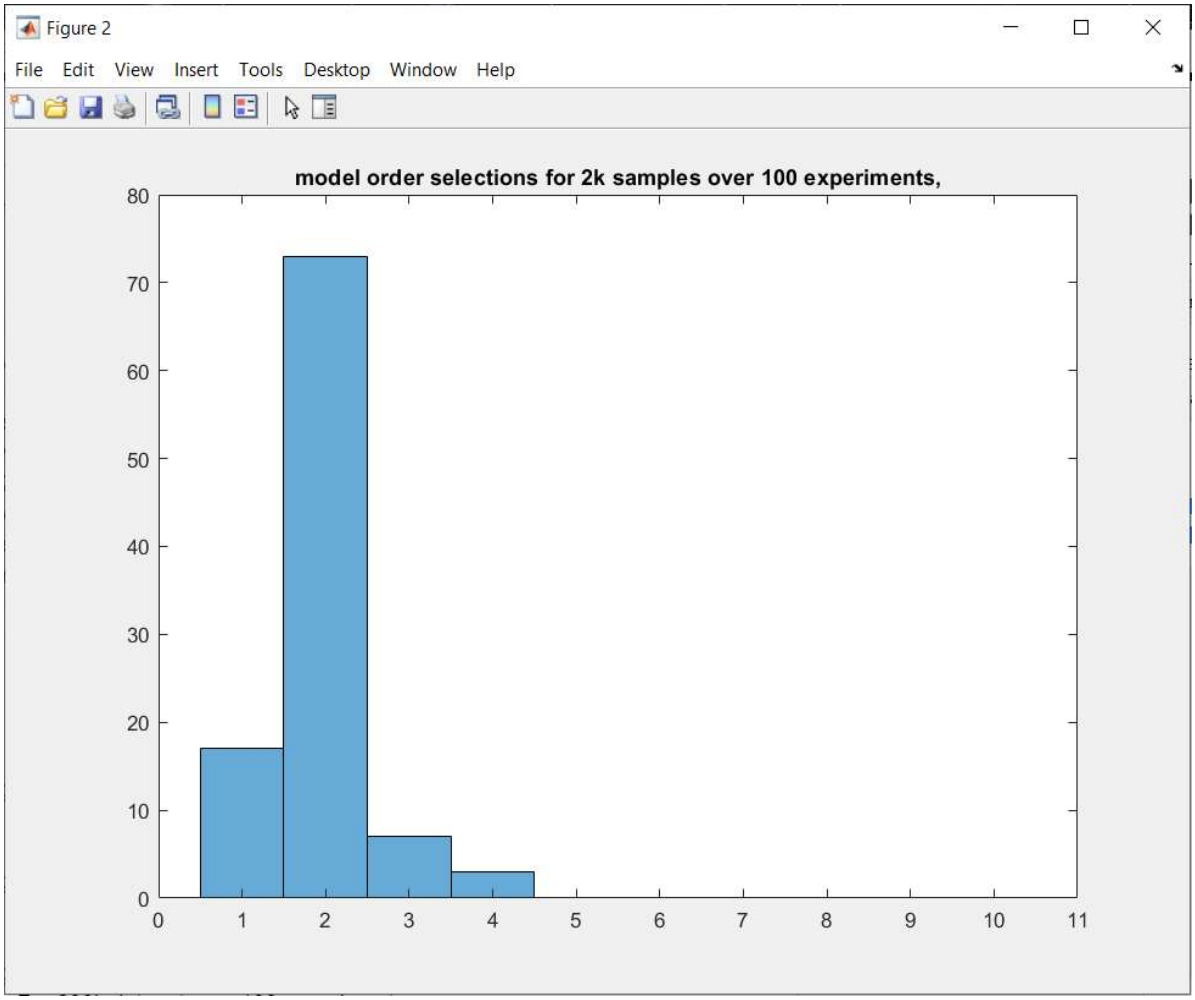

Figure. 12 Histogram of model order selections
For 2k dataset over 100 experiments

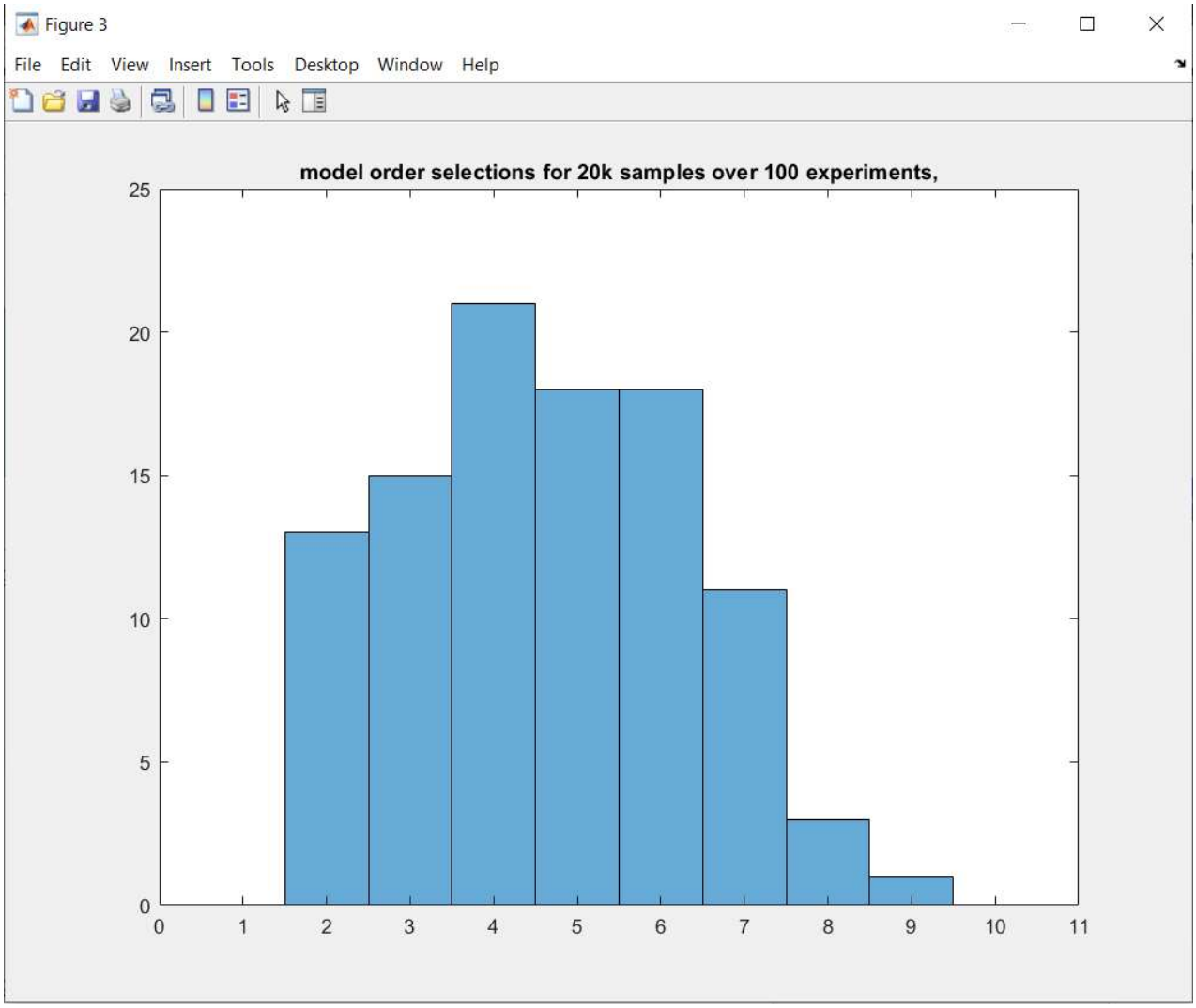

Figure. 13 Histogram of model order selections
For 20k dataset over 100 experiments

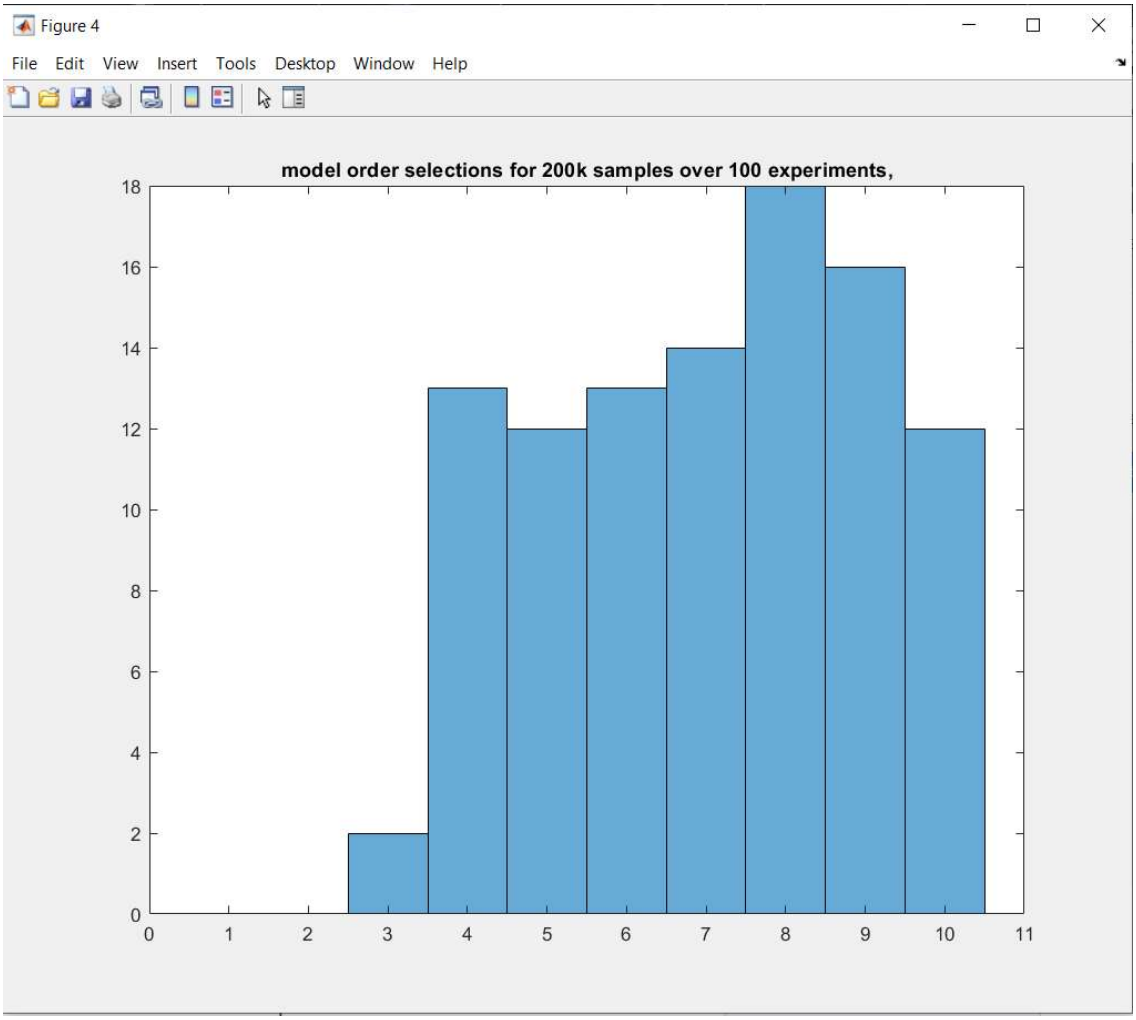

Figure. 14 Histogram of model order selections
For 200k dataset over 100 experiments

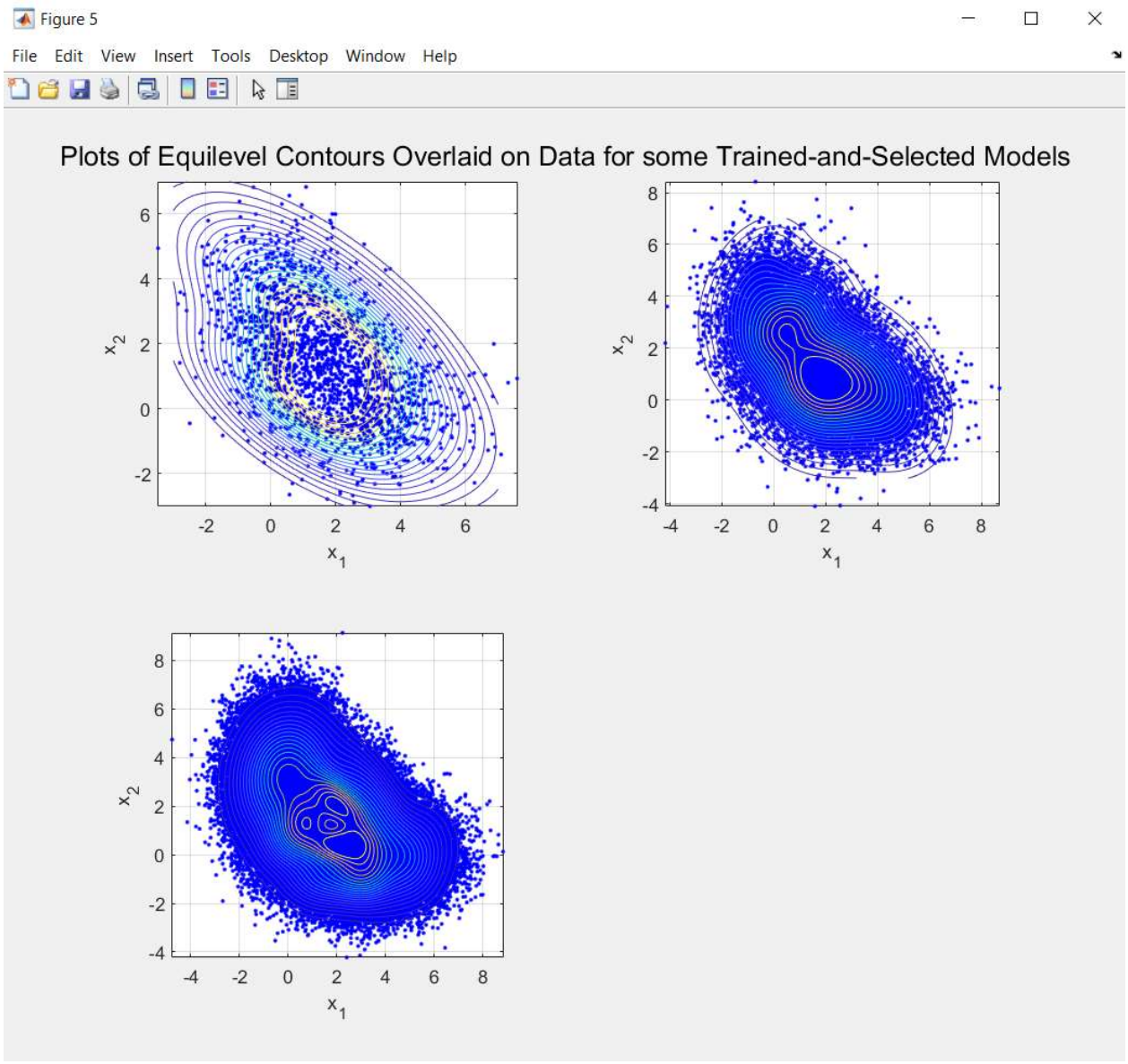

Figure. 15 sample plots of equilevel contours overlaid on data for trained-and-selected models

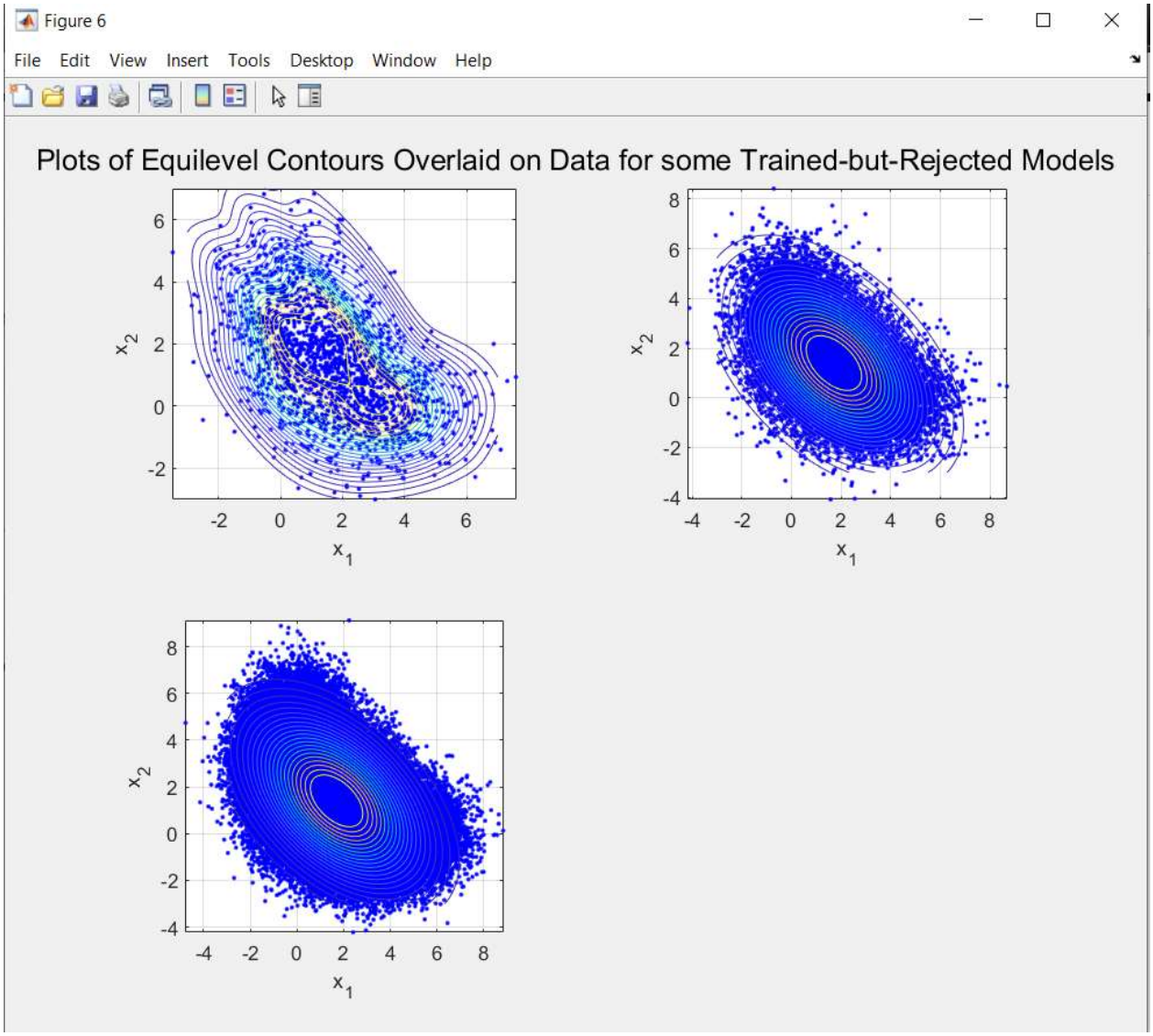

Figure. 16 sample plots of equilevel contours overlaid on data for trained-but-rejected models

Citations

The format of the variables and the code was based on the solution for exam 1 question 3 from summer 1 2019. Moreover, the ROC estimation function and some pieces of generic code (like plotting, probabilities calculations, and basic equations) were copied from the same code.

The GMM ML algorithm (called MLEGMM in the code) and evalGMM function (called evalgmm in this code) was extracted from MLforGMM.m code found in the G/drive. Moreover, the function to plot the gaussian contours was given by the professor during office hours.

Code can be found in g/drive
https://drive.google.com/drive/u/1/folders/1z8xTQoF4b07dwwHcF-m45HDsi3mrYIYq file Exam1_Q3.m and MLforGMM.m

Bilmes, Jeff A. "A Gentle Tutorial of the EM Algorithm and its Application to Parameter Estimation for

Appendix A
Code for Question 1

```
close all, clear all

p1 = 0.65; p2 = 0.35;
alpha = [0.5 0.5];
mu1 = [3 0; 0 3];
sigma1(:,:,1) = [2 0;0 1];
sigma1(:,:,2) = [1 0;0 2];
mu2 = [2;2]; sigma2 = eye(2);

[D50,label50,Nc50] = GenDataQ1(50,1);
[D500,label500,Nc500] = GenDataQ1(500,2);
[D5k,label5k,Nc5k] = GenDataQ1(5000,3);
[D10k,label10k,Nc10k] = GenDataQ1(10000,4);

%Part 1 MAP
disp("Part 1");
Validate(p1,p2,alpha,mu1,sigma1,mu2,sigma2,D10k,label10k,Nc10k,2);

%PArt 2.a
disp('part B.a');
x1 = (D5k(find(label5k == 1),:))';
x2 = (D5k(find(label5k == 2),:))';
[Ealpha1, Emu1, Esigma1] = MLEGMM(x1,3e-2);
[Emu2,Esigma2] = GMLE(x2);
Ealpha1 = Ealpha1';
Validate(p1,p2,Ealpha1,Emu1,Esigma1,Emu2,Esigma2,D10k,label10k,Nc10k,4);
```

```
%Part 2.b
disp('part B.b');
x1 = (D500(find(label500 == 1),:))';
x2 = (D500(find(label500 == 2),:))';
[Ealpha1, Emu1, Esigma1] = MLEGMM(x1,7.5e-2);
[Emu2,Esigma2] = GMLE(x2);
Ealpha1 = Ealpha1';
Validate(p1,p2,Ealpha1,Emu1,Esigma1,Emu2,Esigma2,D10k,label10k,Nc10k,5);

%Part 2.c

disp('part B.c');
x1 = (D50(find(label50 == 1),:))';
x2 = (D50(find(label50 == 2),:))';
[Ealpha1, Emu1, Esigma1] = MLEGMM(x1,4e-1);
[Emu2,Esigma2] = GMLE(x2);
Ealpha1 = Ealpha1';
Validate(p1,p2,Ealpha1,Emu1,Esigma1,Emu2,Esigma2,D10k,label10k,Nc10k,6);

%Part 3.a
%logistic linear function
%training set of 50 samples
disp('50 samples');
LLmodel(D50,50,label50,D10k,label10k,Nc10k);
%training set of 500 samples
disp('500 samples');
LLmodel(D500,500,label500,D10k,label10k,Nc10k);
%training set of 5k samples
disp('5k samples');
LLmodel(D5k,5000,label5k,D10k,label10k,Nc10k);

%Part 3.b
%logistic quadratic function
disp('Logistic-quadratic-function-based');
%training set of 50 samples
disp('50 samples');
LQmodel(D50,50,label50,D10k,label10k,Nc10k);
%training set of 500 samples
disp('500 samples');
LQmodel(D500,500,label500,D10k,label10k,Nc10k);
%training set of 5k samples
disp('5k samples');
```

```
LQmodel(D5k,5000,label5k,D10k,label10k,Nc10k);

%function to implement the logistic linear aproximation
function LLmodel(x_train,N_train,label_train,D10k,label10k,Nc10k)

    %initialize the parameter, make the b(x) vector, define the function
    %and optimize it
    bx = [ones(1,N_train); x_train(:,1)'; x_train(:,2)'];
    w0 = zeros(3,1);
    MinF = @(w)(-1/N_train)*sum((label_train-1).*log(1./(1+exp(-w'*bx)))+
(1-(label_train-1)).*log(1-(1./(1+exp(-w'*bx)))));
    options = optimset('MaxFunEvals', 1e5*6);
    w = fminsearch(MinF,w0,options);

    %validation using the 10k sample data
    bx10k = [ones(1,10000); D10k(:,1)'; D10k(:,2)'];
    H = 1./(1+exp(-w'*bx10k));
    scores = [1-H; H];
    [~,decision] = max(scores,[],1);

    %find min probability of error
    ind11 = find(decision==1 & label10k==1); p11 = length(ind11)/Nc10k(1);
% probability of true negative
    ind21 = find(decision==2 & label10k==1); p21 = length(ind21)/Nc10k(1);
% probability of false positive
    ind12 = find(decision==1 & label10k==2); p12 = length(ind12)/Nc10k(2);
% probability of false negative
    ind22 = find(decision==2 & label10k==2); p22 = length(ind22)/Nc10k(2);
% probability of true positive
    TotalPError = p21*0.65+p12*0.35;
    disp(strcat('min-P error = ',num2str(TotalPError)));
end

%function to implement the logistic linear aproximation
function LQmodel(x_train,N_train,label_train,D10k,label10k,Nc10k)

    %initialize the parameter, make the b(x) vector, define the function
    %and optimize it
    bx = [ones(1,N_train); x_train(:,1)';
x_train(:,2)';(x_train(:,1)').^2;(x_train(:,1)').*(x_train(:,2)');(x_train(
:,2)').^2];
    w0 = zeros(6,1);
```

```
    MinF =
@(w)(-1/N_train)*sum((label_train-1).*log(1./(1+exp(-w'*bx)))+(1-(label_tra
in-1)).*log(1-(1./(1+exp(-w'*bx)))));
    options = optimset('MaxFunEvals', 1e5*6);
    w = fminsearch(MinF,w0,options);

    %validation using the 10k sample data
    bx10k = [ones(1,10000); D10k(:,1)';
D10k(:,2)';(D10k(:,1)').^2;D10k(:,1)'.*D10k(:,2)';(D10k(:,2)').^2];
    H = 1./(1+exp(-w'*bx10k));
    scores = [1-H; H];
    [~,decision] = max(scores,[],1);

    %find min probability of error
    ind11 = find(decision==1 & label10k==1); p11 = length(ind11)/Nc10k(1);
% probability of true negative
    ind21 = find(decision==2 & label10k==1); p21 = length(ind21)/Nc10k(1);
% probability of false positive
    ind12 = find(decision==1 & label10k==2); p12 = length(ind12)/Nc10k(2);
% probability of false negative
    ind22 = find(decision==2 & label10k==2); p22 = length(ind22)/Nc10k(2);
% probability of true positive
    TotalPError1 = p21*0.65+p12*0.35;
    disp(strcat('min-P error = ',num2str(TotalPError1)));
end
%function of ML algorithm (taken from EMforGMM.m)
function [Ealpha1, Emu1, Esigma1] = MLEGMM(x1,delta)

    converged = 0; M = 2; d = 2; regWeight = 1e-10; N = length(x1);

    Ealpha1 = ones(1,M)/M;
    shuffledIndices = randperm(N);
    Emu1 = x1(:,shuffledIndices(1:M)); % pick M random samples as initial
mean estimates
    [~,assignedCentroidLabels] = min(pdist2(Emu1',x1'),[],1); % assign each
sample to the nearest mean
    for m = 1:M % use sample covariances of initial assignments as initial
covariance estimates
        Esigma1(:,:,m) = cov(x1(:,find(assignedCentroidLabels==m))') +
regWeight*eye(d,d);
    end
    t = 0; %displayProgress(t,x,alpha,mu,Sigma);
```

```
    while ~converged & t <=500

        for l = 1:M
            temp(l,:) =
repmat(Ealpha1(l),1,N).*(mvnpdf(x1',Emu1(:,l)',Esigma1(:,:,l)))';
        end

        plgivenx = temp./sum(temp,1);
        alphaNew = mean(plgivenx,2);
        w = plgivenx./repmat(sum(plgivenx,2),1,N);
        muNew = x1*w';
        for l = 1:M
            v = x1-repmat(muNew(:,l),1,N);
            u = repmat(w(l,:),d,1).*v;
            SigmaNew(:,:,l) = u*v' + regWeight*eye(d,d); % adding a small
regularization term
        end
        Dalpha = sum(abs(alphaNew-Ealpha1'));
        Dmu = sum(sum(abs(muNew-Emu1)));
        DSigma = sum(sum(abs(abs(SigmaNew-Esigma1))));
        converged = ((Dalpha+Dmu+DSigma)<delta); % Check if converged
        Ealpha1 = alphaNew; Emu1 = muNew; Esigma1 = SigmaNew;
        t = t+1;
    end
end
% function with sample average and covariance estimator
function [Emu2,Esigma2] = GMLE(x2)
    N = length(x2);
    Emu2 = mean(x2,2);
    Esigma2 = cov(x2');
end

%function to validate using the 10k sample data set and the MAP desicion
rule
function Validate(p1,p2,alpha,mu1,sigma1,mu2,sigma2, D10k,label10k,Nc10k,i)
score1 = (mvnpdf(D10k,mu1(:,1)',sigma1(:,:,1))*alpha(1) +
mvnpdf(D10k,mu1(:,2)',sigma1(:,:,2))*alpha(2))*p1;
score2 = mvnpdf(D10k,mu2',sigma2)*p2;
Scores = [score1 score2];

[~,decision] = max((log(Scores))');
```

```
ind11 = find(decision==1 & label10k==1); p11 = length(ind11)/Nc10k(1); %
probability of true negative
ind21 = find(decision==2 & label10k==1); p21 = length(ind21)/Nc10k(1); %
probability of false positive
ind12 = find(decision==1 & label10k==2); p12 = length(ind12)/Nc10k(2); %
probability of false negative
ind22 = find(decision==2 & label10k==2); p22 = length(ind22)/Nc10k(2); %
probability of true positive

%MAP ROC Curve
discriminantScore = (log(score1./score2))';
[ROC,tau] = estimateROC(discriminantScore,label10k,Nc10k);
probError = [((ROC(1,:).*p1)+(ROC(3,:).*p2))]; % probability of total error
for different threshold values
[pEmin,ind] = min(probError);
plotROC(ROC,ind,i)

TotalPError1 = p21*p1+p12*p2;
disp(strcat('Theoretical min-P error = ',num2str(TotalPError1)));
disp(strcat('ROC min-P error = ',num2str(pEmin)));
end

%ROC estimation nd plottin function (taken from Exam1_Q1.m summe1 2019)
function [ROC,tau] = estimateROC(discriminantScore,label,Nc)
% Generate ROC curve samples
sortedScore = sort(discriminantScore,'ascend');
tau =
[sortedScore(1),(sortedScore(2:end)+sortedScore(1:end-1))/2,sortedScore(end
)+1];
%thresholds at midpoints of consecutive scores in sorted list
for k = 1:length(tau)
    decision = (discriminantScore <= tau(k))+1;
    ind21 = find(decision==2 & label==1); p21 = length(ind21)/Nc(1); %
probability of false positive
    ind22 = find(decision==2 & label==2); p22 = length(ind22)/Nc(2); %
probability of true positive
    ind12 = find(decision==1 & label==2); p12 = length(ind12)/Nc(2); %
probability of false negative
    ROC(:,k) = [p21;p22;p12];
end
end
```

```
function plotROC(ROC,ind,i)
% Display the estimated ROC curve with blue o
% and indicate the  min-P(error) classifier with a red +
figure(i), plot(ROC(1,:),ROC(2,:),'bo'); hold on,
plot(ROC(1,ind),ROC(2,ind),'r+');
axis equal, xlim([0,1]); ylim([0,1]), xlabel('P False Positive'),ylabel('P
True Positive');
title('ROC Curve');
end

%functions to generate and plot the data sets
function [x,label,Nc] = GenDataQ1(N,i)
p1 = 0.65; p2 = 0.35;
alpha = [0.5 0.5];
mu1 = [3 0; 0 3];
sigma1(:,:,1) = [2 0;0 1];
sigma1(:,:,2) = [1 0;0 2];
mu2 = [2;2]; sigma2 = eye(2);

u = rand(1,N)>= p1; N1 = length(find(u==0)); N2 = length(find(u==1)); Nc =
[N1 N2];
x0 = (randGMM(N1,alpha,mu1,sigma1))';
x1 = mvnrnd(mu2, sigma2, N2);
x = [x0 ; x1];
label = [ones(1,N1) 2*ones(1,N2)];
plotData(x0,x1,i);
end

function x = 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) = (mvnrnd(mu(:,m), Sigma(:,:,m), length(ind)))';
end
end

function plotData(x0,x1,i)
figure(1),sgtitle('Training Data 50 500 5k 10k samples');
figure(1),subplot(2,2,i), plot(x0(:,1),x0(:,2),'.b'); axis equal, hold on;
```

```
figure(1),subplot(2,2,i), plot(x1(:,1),x1(:,2),'.r'); axis equal,hold on;
xlabel('x1'),ylabel('x2');
end
```

Appendix B

Code for Question 2

```
close all, clear all

count2k = zeros(1,10); count20k = zeros(1,10); count200k = zeros(1,10);

for i = 1:150
    D2k = (GenDataQ2(2000,1))';
    D20k = (GenDataQ2(20000,2))';
    D200k = (GenDataQ2(200000,3))';

    decision2k = BIC(D2k,2000);
    count2k(decision2k) =  count2k(decision2k) + 1;
    i
    decision20k = BIC(D20k,20000);
    count20k(decision20k) =  count20k(decision20k) + 1;
    i
    decision200k = BIC(D200k,200000);
    count200k(decision200k) =  count200k(decision200k) + 1;
    i
end
plotData(D2k',1,1); plotData(D20k',2,1); plotData(D200k',3,1);

figure (2), histogram('BinEdges',[0.5:10.5],'BinCounts',count2k),
title('model order selections for 2k samples over 100 experiments,')
figure (3), histogram('BinEdges',[0.5:10.5],'BinCounts',count20k),
title('model order selections for 20k samples over 100 experiments,')
figure (4), histogram('BinEdges',[0.5:10.5],'BinCounts',count200k),
title('model order selections for 200k samples over 100 experiments,')

plotData(D2k',1,5); plotData(D20k',2,5); plotData(D200k',3,5);
figure(5), clear sgtitle, sgtitle('Plots of Equilevel Contours Overlaid on
```

```
Data for some Trained-and-Selected Models')

[~,Max] = max(count2k)
[Ealpha2k, Emu2k, Esigma2k] = MLEGMM(D2k,1e-1,Max);
cont(Emu2k,Esigma2k,Ealpha2k,1,5);
[~,Max] = max(count20k)
[Ealpha20k, Emu20k, Esigma20k] = MLEGMM(D20k,1e-1,Max);
cont(Emu20k,Esigma20k,Ealpha20k,2,5);
[~,Max] = max(count200k)
[Ealpha200k, Emu200k, Esigma200k] = MLEGMM(D200k,1e-1,Max);
cont(Emu200k,Esigma200k,Ealpha200k,3,5);

plotData(D2k',1,6); plotData(D20k',2,6); plotData(D200k',3,6);
figure(6), clear sgtitle, sgtitle('Plots of Equilevel Contours Overlaid on
Data for some Trained-but-Rejected Models')
[~,Min] = min(count2k)
[Ealpha2k, Emu2k, Esigma2k] = MLEGMM(D2k,1e-1,Min);
cont(Emu2k,Esigma2k,Ealpha2k,1,6);
[~,Min] = min(count20k)
[Ealpha20k, Emu20k, Esigma20k] = MLEGMM(D20k,1e-1,Min);
cont(Emu20k,Esigma20k,Ealpha20k,2,6);
[~,Min] = min(count200k)
[Ealpha200k, Emu200k, Esigma200k] = MLEGMM(D200k,1e-1,Min);
cont(Emu200k,Esigma200k,Ealpha200k,3,6);

function decision = BIC(x,N)
    for i = 1:10
        [Ealpha, Emu, Esigma] = MLEGMM(x,8e-1,i);
        BIC(:,i) =
-2*sum(log(evalgmm(x,Emu,Esigma,Ealpha,i)))+(6*i-1)*log(2*N);
    end
    %figure(2), plot(BIC)
    [~,decision] = min(BIC);
end

function [Ealpha, Emu, Esigma] = MLEGMM(x,delta,M)

    converged = 0; d = 2; regWeight = 1e-10; N = length(x);

    Ealpha = ones(1,M)/M;
    shuffledIndices = randperm(N);
    Emu = x(:,shuffledIndices(1:M)); % pick M random samples as initial
```

```
mean estimates
    [~,assignedCentroidLabels] = min(pdist2(Emu',x'),[],1); % assign each
sample to the nearest mean
    for m = 1:M % use sample covariances of initial assignments as initial
covariance estimates
        Esigma(:,:,m) = cov(x(:,find(assignedCentroidLabels==m))') +
regWeight*eye(d,d);
    end
    t = 0; %displayProgress(t,x,alpha,mu,Sigma);

    while (~converged) & (t <= 300)

        for l = 1:M
            temp(l,:) =
repmat(Ealpha(l),1,N).*(mvnpdf(x',Emu(:,l)',Esigma(:,:,l)))';
        end

        plgivenx = temp./sum(temp,1);
        alphaNew = mean(plgivenx,2);
        w = plgivenx./repmat(sum(plgivenx,2),1,N);
        muNew = x*w';
        for l = 1:M
            v = x-repmat(muNew(:,l),1,N);
            u = repmat(w(l,:),d,1).*v;
            SigmaNew(:,:,l) = u*v' + regWeight*eye(d,d); % adding a small
regularization term
        end
        Dalpha = sum(abs(alphaNew-Ealpha'));
        Dmu = sum(sum(abs(muNew-Emu)));
        DSigma = sum(sum(abs(abs(SigmaNew-Esigma))));
        converged = ((Dalpha+Dmu+DSigma)<delta); % Check if converged
        Ealpha = alphaNew; Emu = muNew; Esigma = SigmaNew;
        t = t+1;
        %displayProgress(t,x,Ealpha,Emu,Esigma,M);
    end
end

function plotData(x0,i,i2)
figure(i2),sgtitle('Training Data 2k 20k 200k samples');
figure(i2),subplot(2,2,i), plot(x0(:,1),x0(:,2),'.b'); axis equal, hold on;
xlabel('x1'),ylabel('x2');
end
```

```
function cont(mu,sigma,alpha,i,i2)
figure(i2),subplot(2,2,i),
px0 = gmdistribution(mu',sigma,alpha);

Nx = 101; Ny = 99;
xGrid = linspace(-3,7,Nx); yGrid = linspace(-3,7,Ny);
[h,v] = meshgrid(xGrid,yGrid);
gridPoints = [h(:),v(:)];

pdfGMM = pdf(px0,gridPoints);
zGrid = reshape(pdfGMM,Ny,Nx);
% Display the MAP objective contours
minV = min(pdfGMM); maxV = max(pdfGMM);
values = minV + (sqrt(maxV-minV)*linspace(0.1,0.9,21)).^2;
contour(xGrid,yGrid,zGrid,values); xlabel('x_1'), ylabel('x_2'),
%contour(xGrid,yGrid,zGrid); xlabel('x_1'), ylabel('x_2'),
grid on, axis equal,

end

function x = GenDataQ2(N,i)

alpha = [0.4 0.3 0.2 0.1];
mu(:,:,1) = [3 0];
mu(:,:,2) = [0 3];
mu(:,:,3) = [2 2];
mu(:,:,4) = [1 1];
Sigma(:,:,1) = [2 0;0 1];
Sigma(:,:,2) = [1 0;0 2];
Sigma(:,:,3) = eye(2);
Sigma(:,:,4) = eye(2);

x = (randGMM(N,alpha,mu,Sigma))';
end
function x = randGMM(N,alpha,mu,Sigma)
d = size(mu,2); % 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) = (mvnrnd(mu(:,:,m)', Sigma(:,:,m), length(ind)))';
end
```

```
end
function pdf = evalgmm(x,mu,sigma,alpha,M)
    pdf = zeros(1,size(x,2));
    for i = 1:M % evaluate the GMM on the grid
        pdf = pdf + (mvnpdf(x',mu(:,i)',sigma(:,:,i)))'*alpha(i);
    end
end
```