**Assignment 3**

**DANDAN LIN**

**NUID:001093902**

**Question 1:**

For this question a 3-dimensional real-values random vector x was generated from 4 classes with uniform priors and Gaussian class conditional pdfs. The distributions used are shown below.











MAP KNOWLEDGE:













Apply :



We can konw:

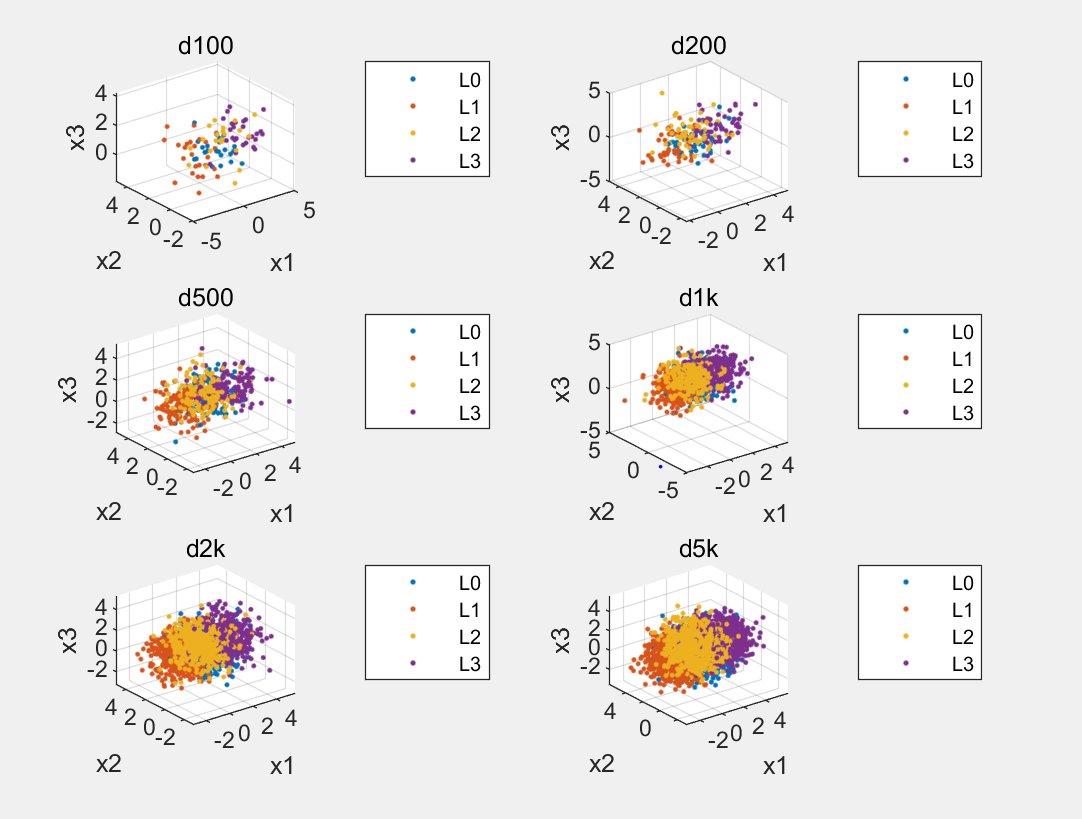
So 



The question 1 asked to use a 2-layer MLP (one hidden layer of perceptrons) that has *P* perceptrons in the first (hidden) layer with smooth-ramp style activation functions (e.g., ISRU, Smooth-ReLU,

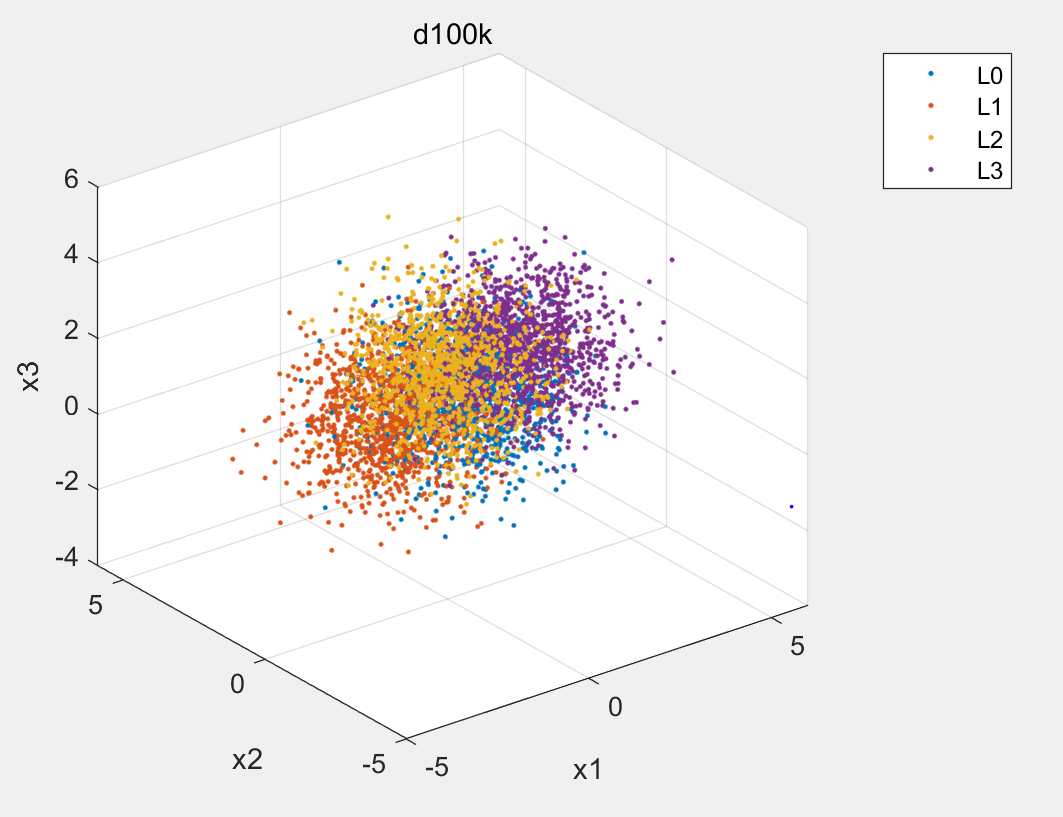
ELU, etc). At the second/output layer use a softmax function to ensure all outputs are positive and add up to 1. and the output layer was a “softmax” function as is the default for the Matlab “patternnet” function, however I do not know use which built-in function of that type for “patternnet” in Matlab at first.After reading the practice example which professor offers to us and use it as reference,I know that in Matlab, we can use default function “tansig” instead.

Figure 1 is for training datasets with 100, 200, 500, 1000, 2000, and 5000 samples .



*Figure 1 training datasets*

Figure 2 is for validation a test dataset with 100,000 samples.

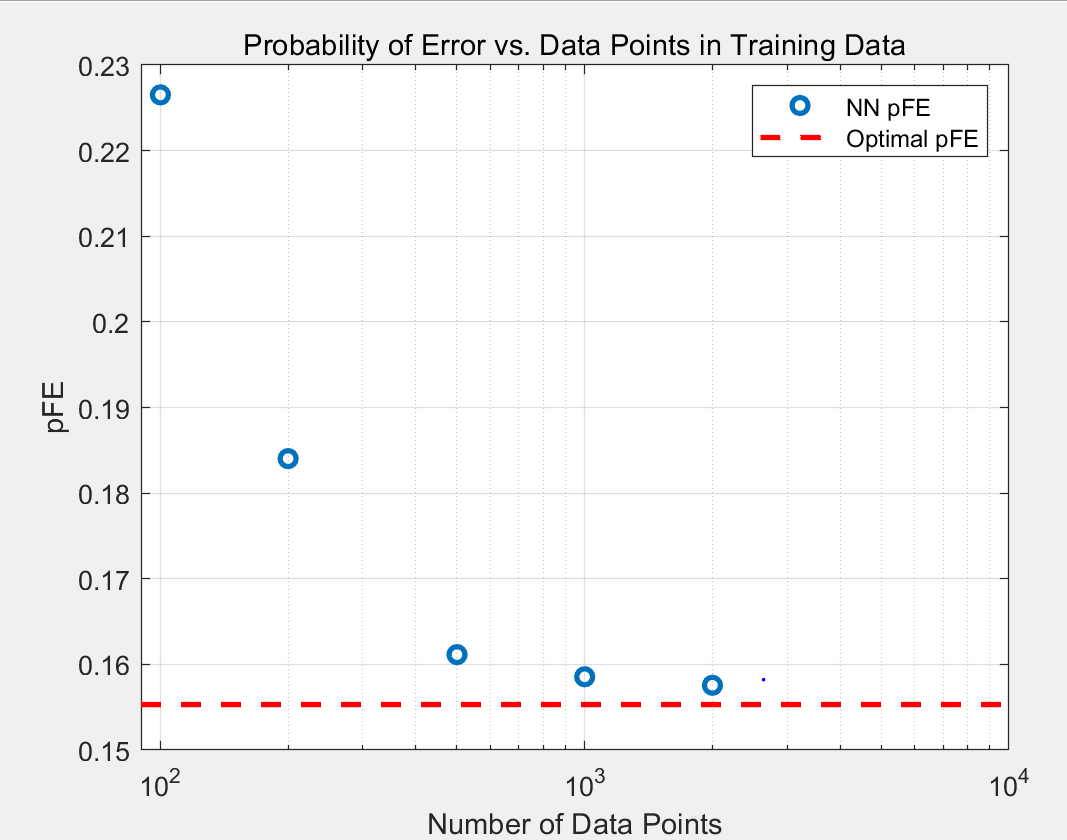


*Figure 2 test datasets*

Using the knowledge of true data pdf, construct the minimum probability of error classification rule ,apply it on the test dataset and finally calculate the theoretically optimal pfe.Then for each training dataset 10-fold cross validation was performed to determine the optimal number of

perceptrons for the MLP model. The optimal number was the one that resulted in the minimum probability of error across the cross validation runs,Which we could seen results from figure3.After we selected the number of perceptrons,we could apply it on the training dataset. Finally, this trained model could compared with the test dataset result and the probability of error was calculated as the standard data.

In the figure below,we could find that as the size of the dataset increases the probability of error decreases and approaches the optimal probability of error as estimated using the true pdf . This indicates that as the number of training data increases,we could get more accurate classifications with the same model.



*Figure 3 pe vs data points*

For figure 4,we could find that overall the optimal number of perceptrons appears to increase as the size of the training dataset increases. Although there may be some data float which was shown in figure,the general trend was still rising.This because as the size of the dataset increases the complexity of the model can also increase. More data means more features than can be modeled and therefore model complexity increases.

*Figure 4 perceptrons vs data points*

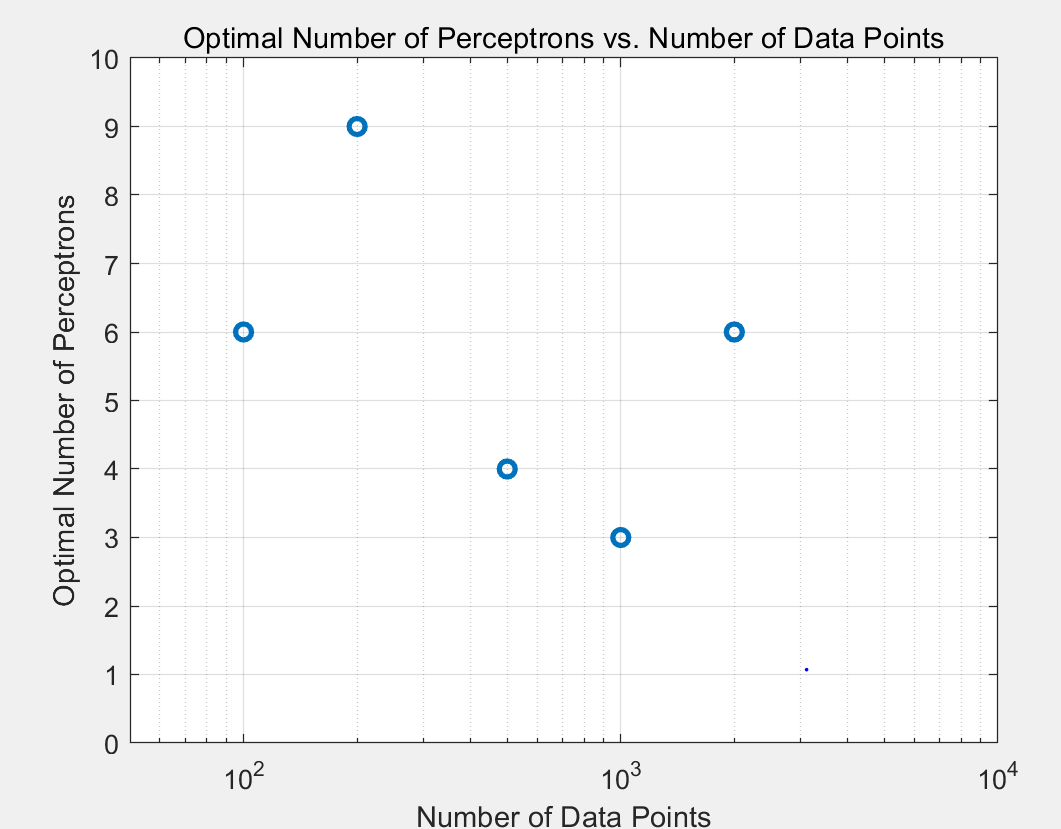
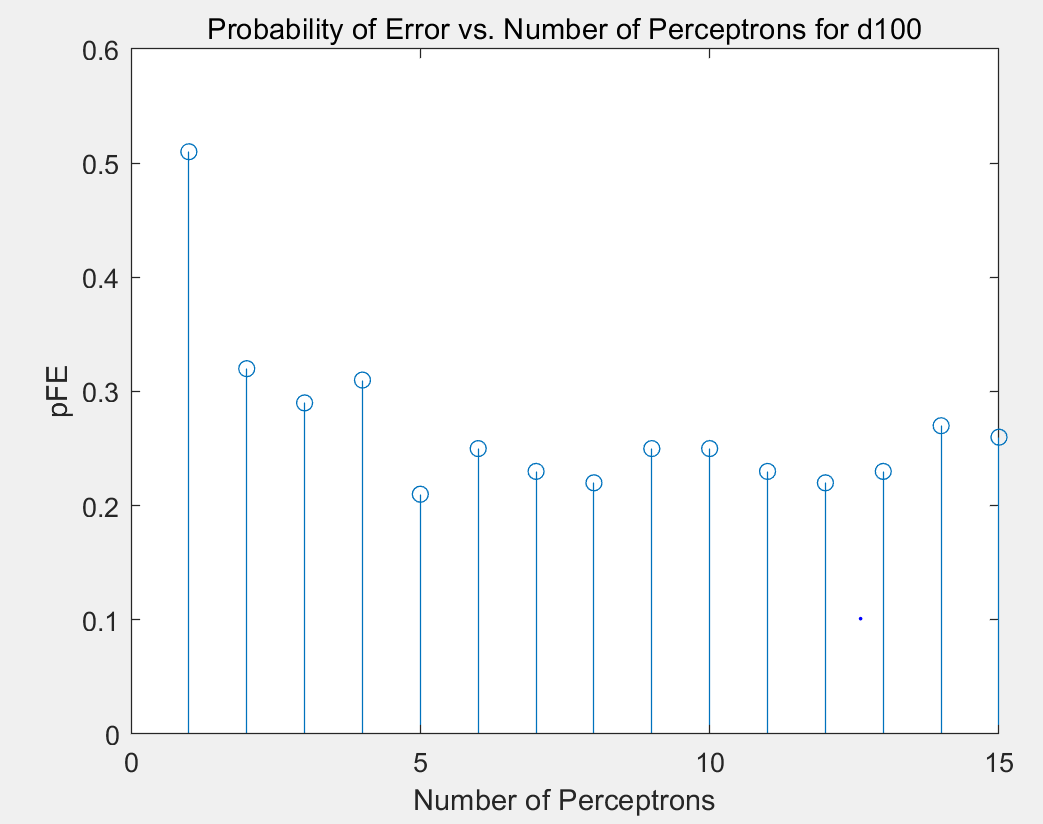


Table 1 shows the optimal and the neutron network minimum probability of error with different datasets.(There is some data float)

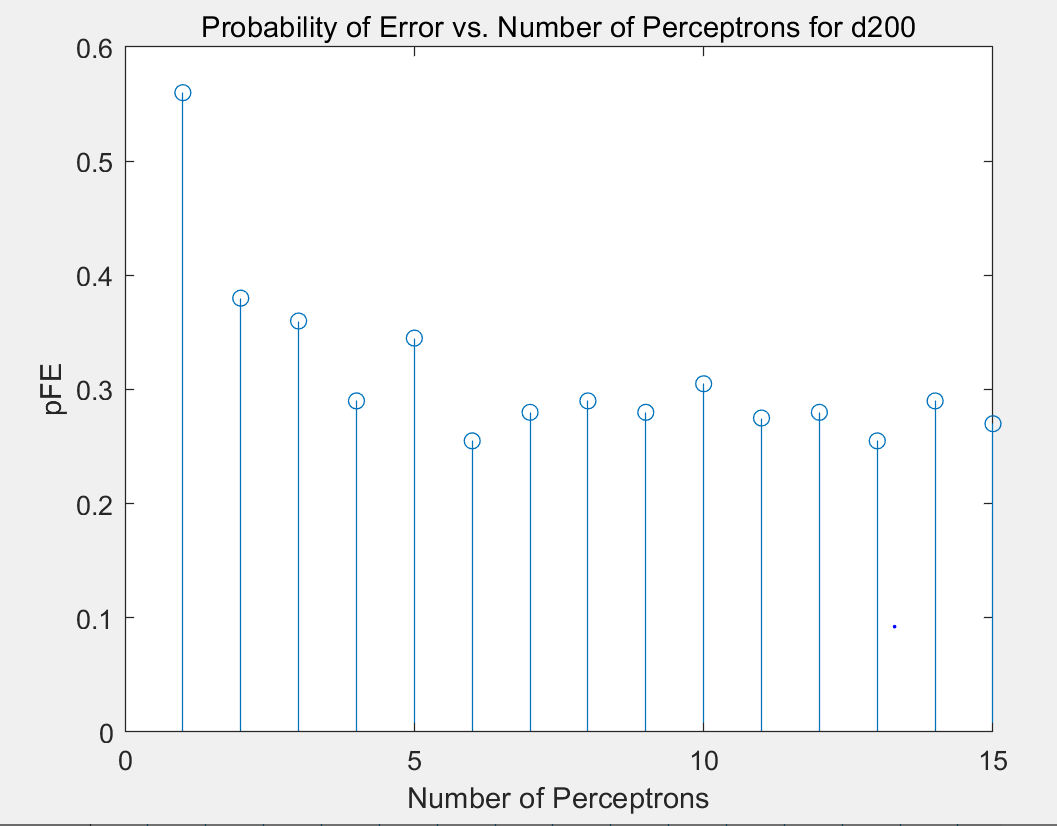
*Table 1: optimal and NN Probability of Error*

|  |  |  |
| --- | --- | --- |
|  | Optimal pe | NN pe |
| 100 | 11.00% | 22.65% |
| 200 | 21.50% | 18.40% |
| 500 | 16.60% | 16.11% |
| 1000 | 14.40% | 15.86% |
| 2000 | 14.55% | 15.76% |
| 5000 | 15.38% | 15.59% |
| 10000 | 15.53% |  |

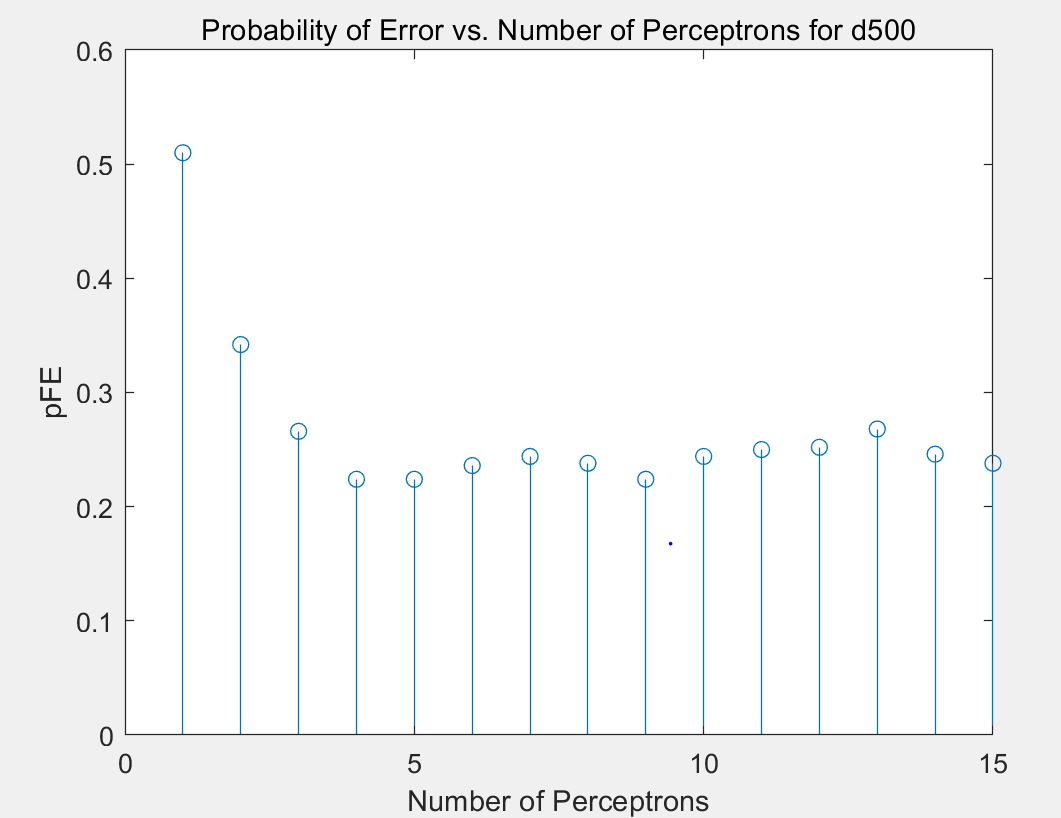
Figure 5-10 below show plots of the cross-validation results for the differernt sample training dataset. In the plot the probability of error correlates with the number of perceptrons. The probability of error starts very large for a single perceptron and then rapidly decreases as the number of perceptrons increases. When they reached the minimum, the change became smaller and then overall the probability error slowly increased as the number of perceptrons increased. While the optimal number of perceptrons varied between training datasets, all plots followed this pattern.



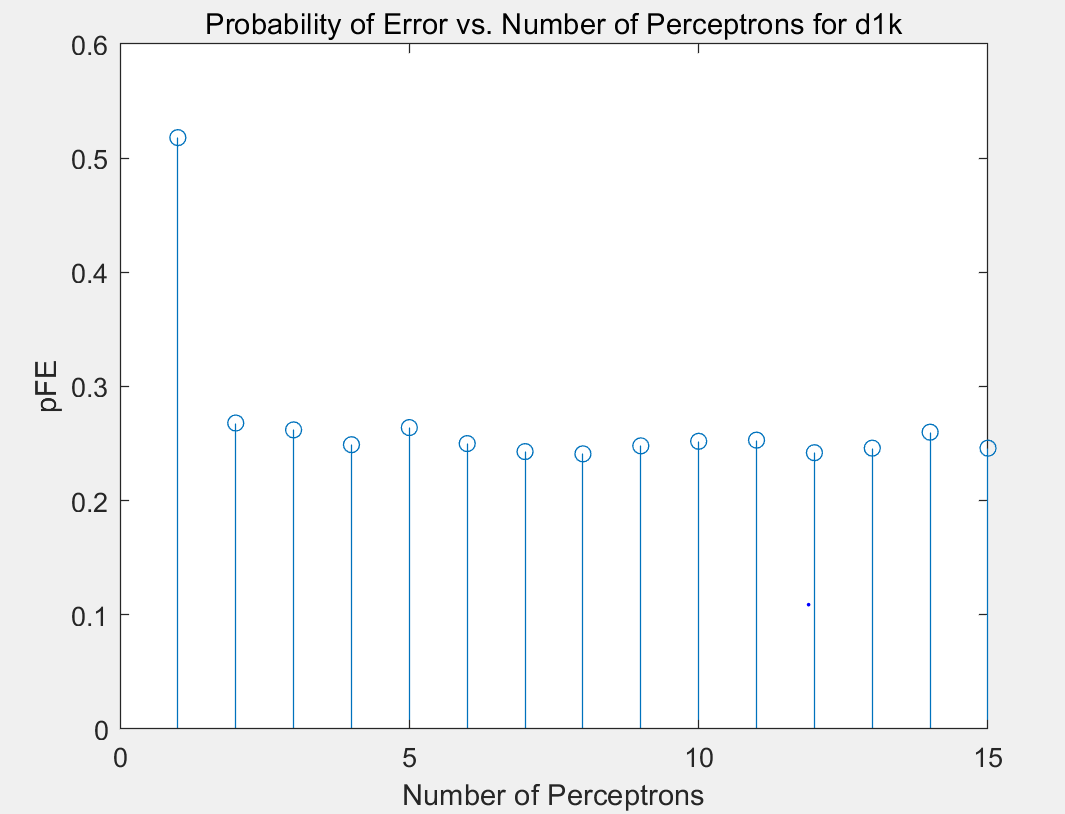
*Figure 5*



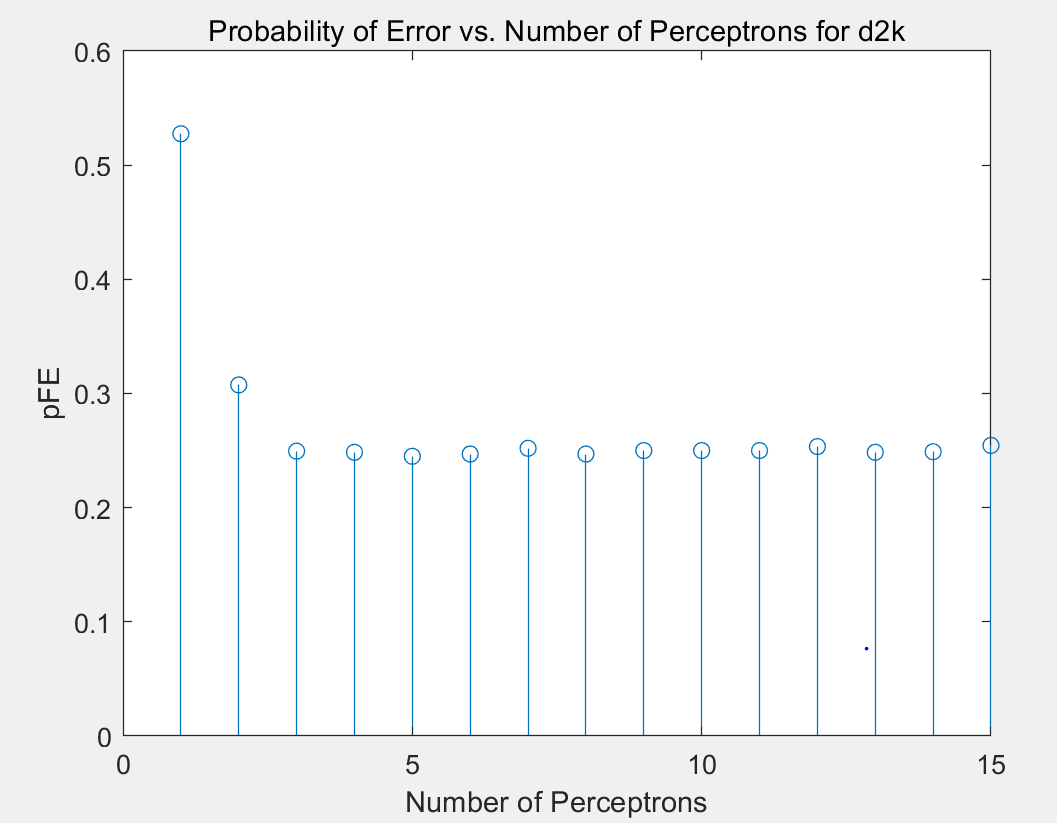
*Figure 6*



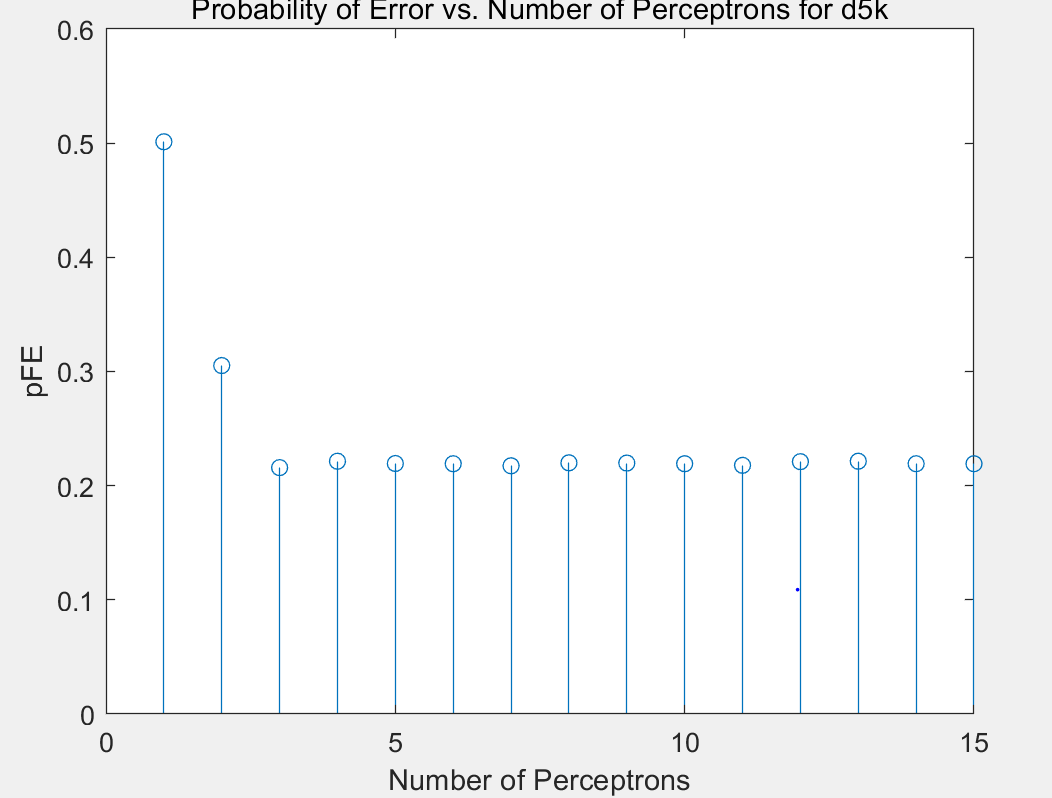
*Figure 7*



*Figure 8*



*Figure 9*



*Figure 10*

**Question 2:**





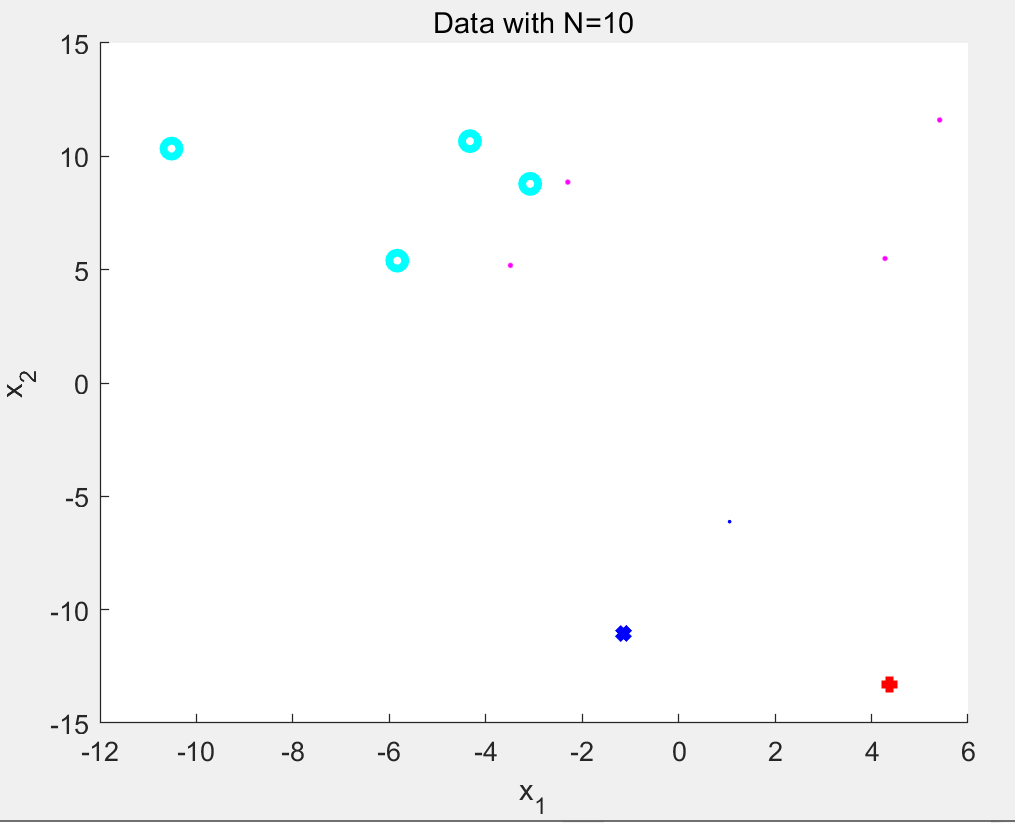




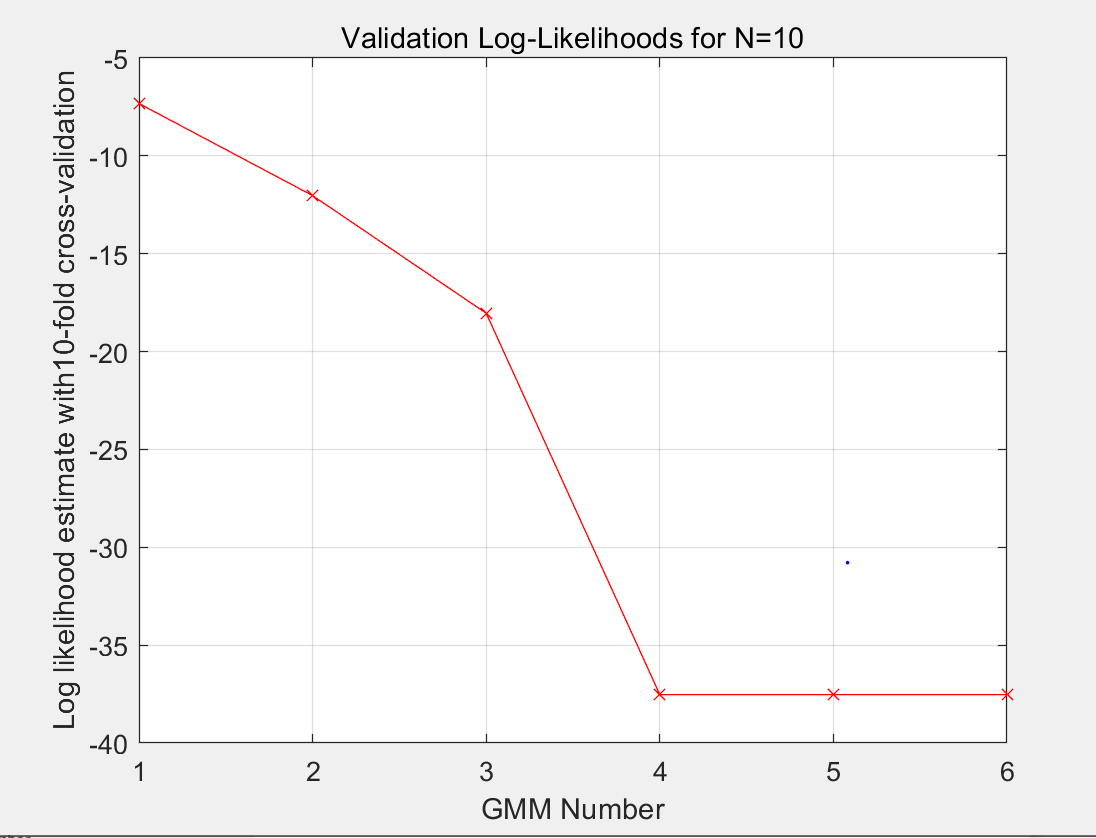
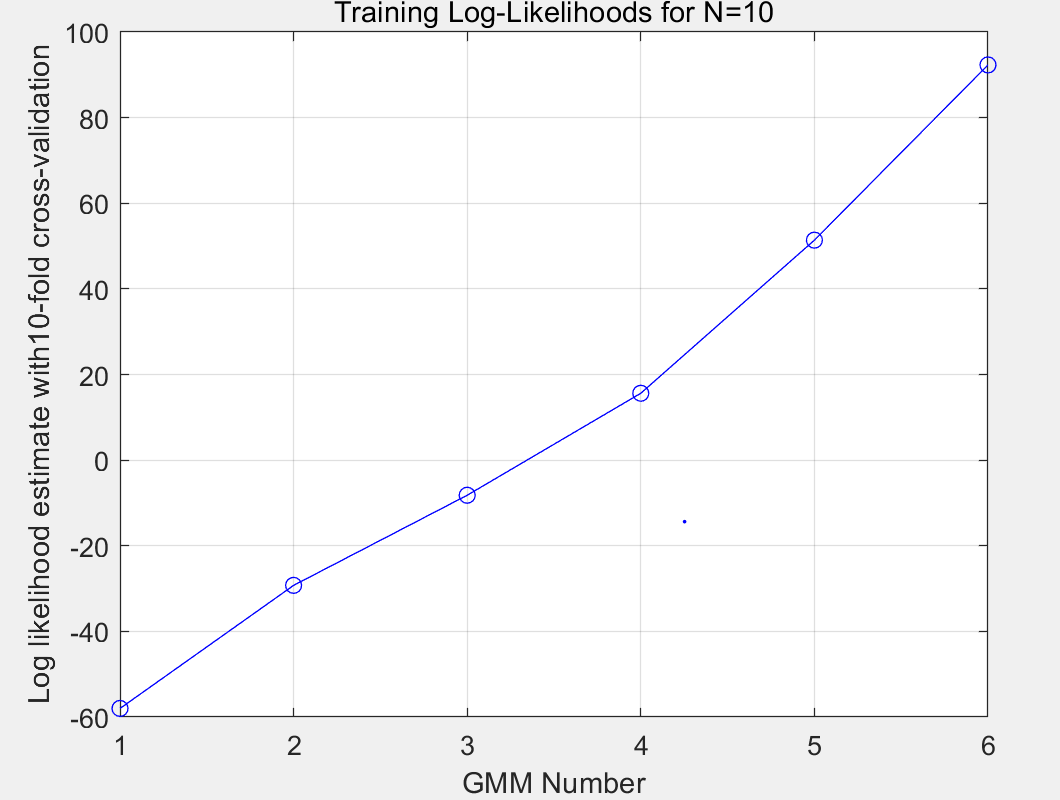


In this question,I use i=1：experiments(experiments=30) loop to finish data training ,results as below:

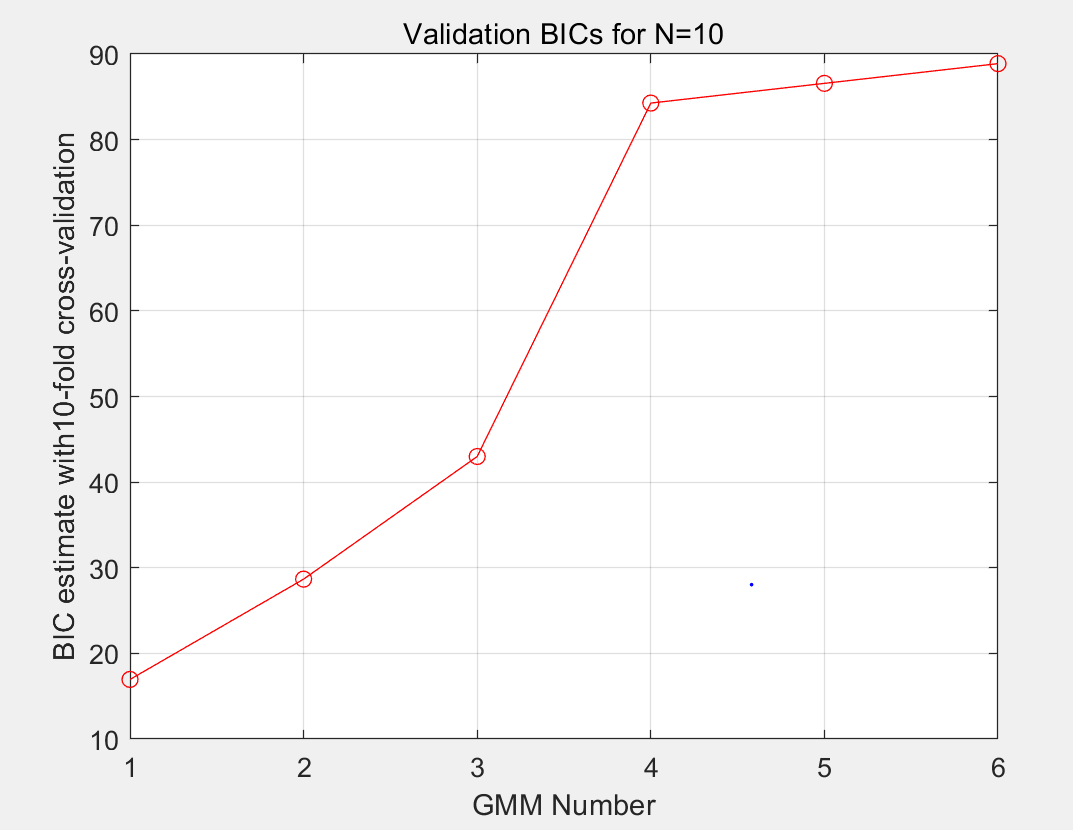
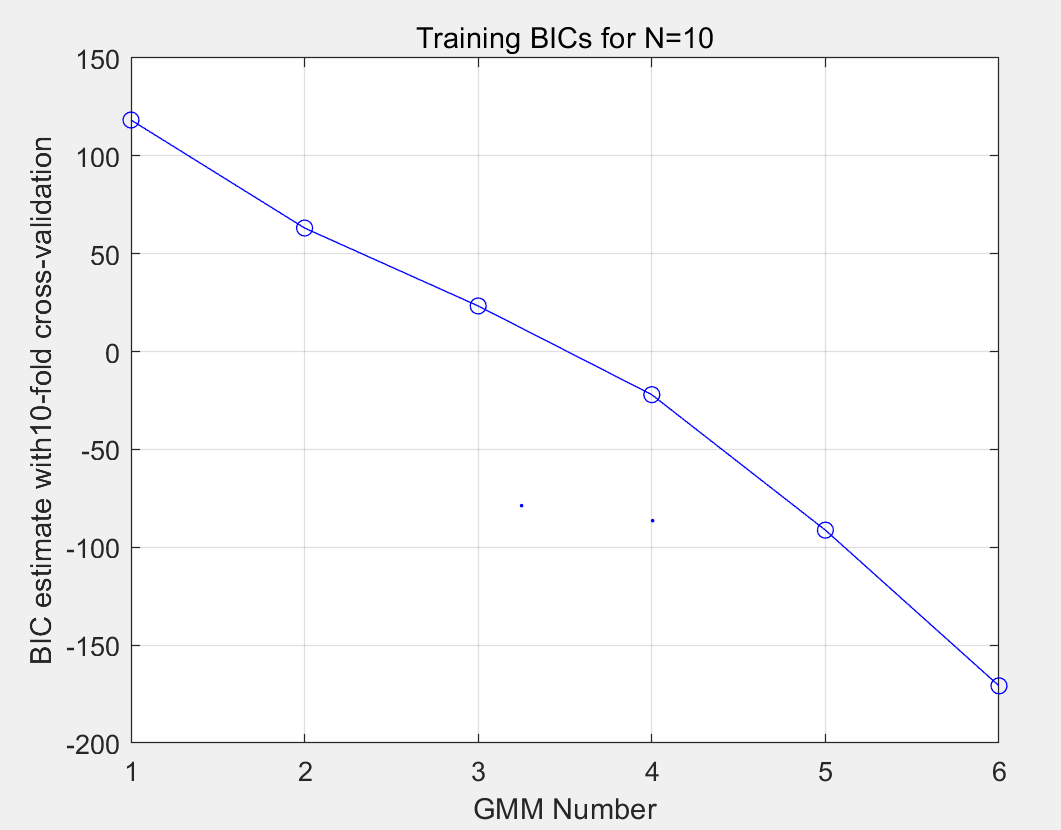
N=10



*Figure 11 4class for N=10*

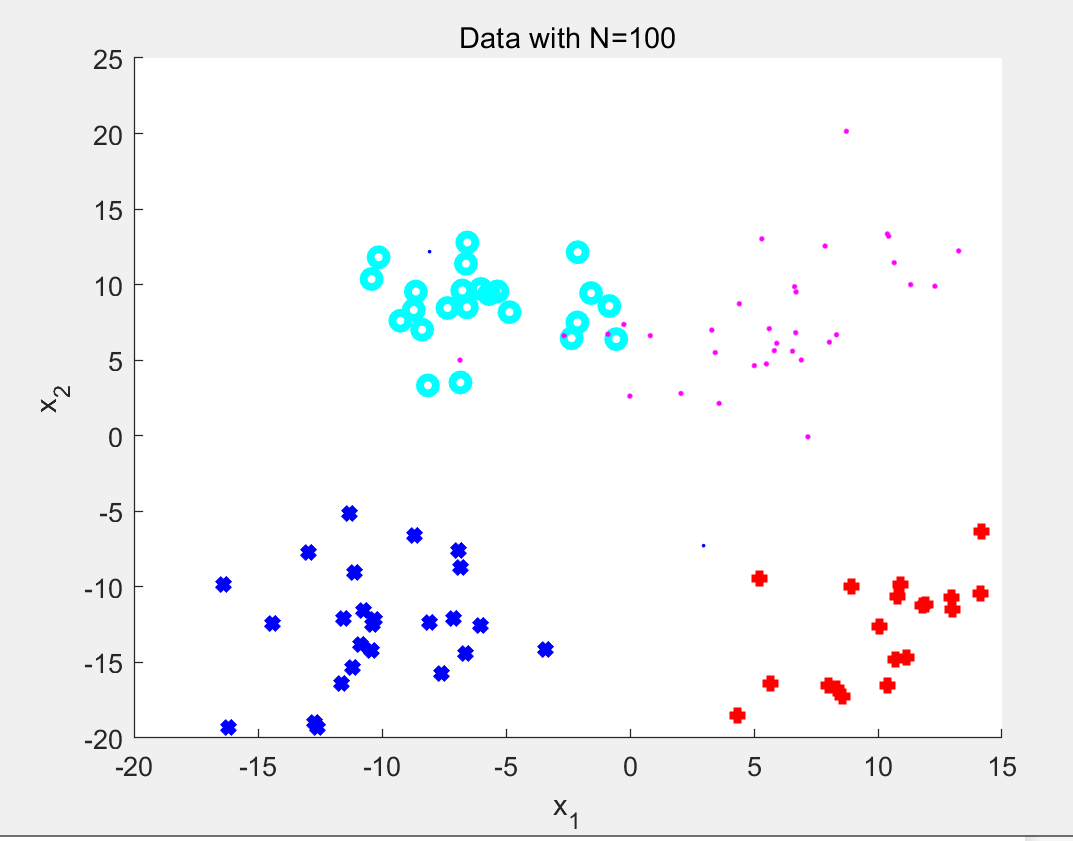


*Figure 12 loglikelihood for N=10*

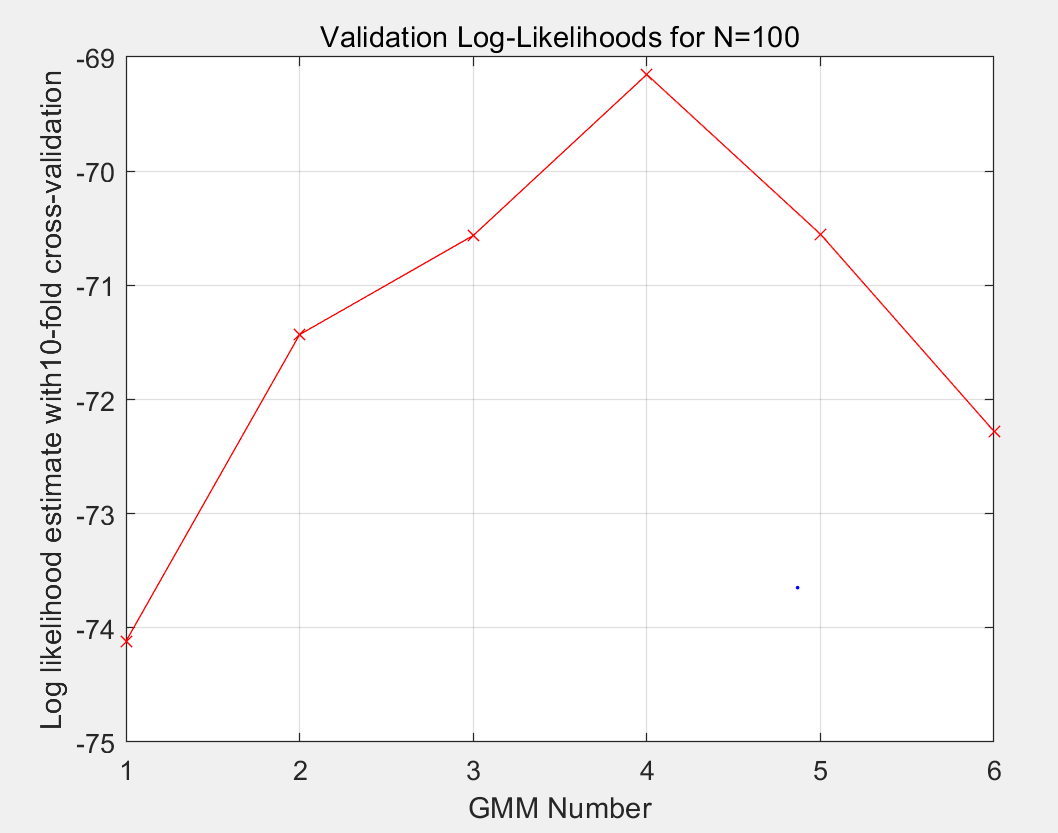
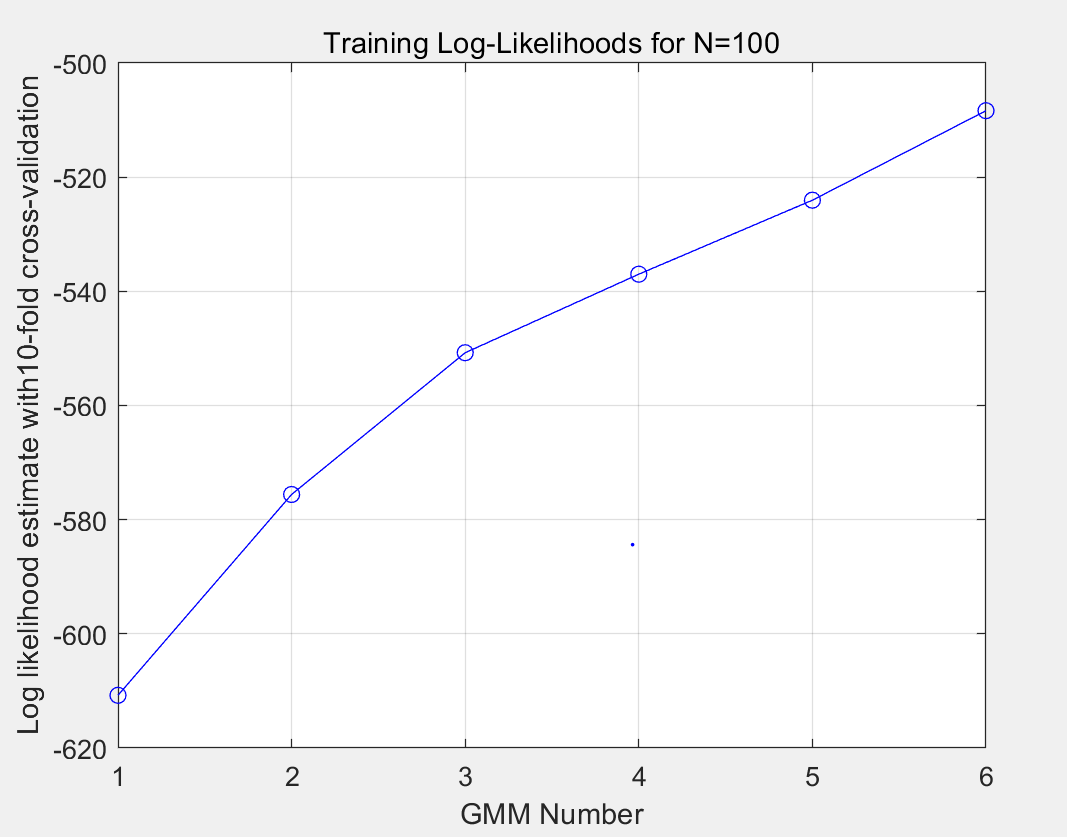


*Figure 13 BIC for N=10*

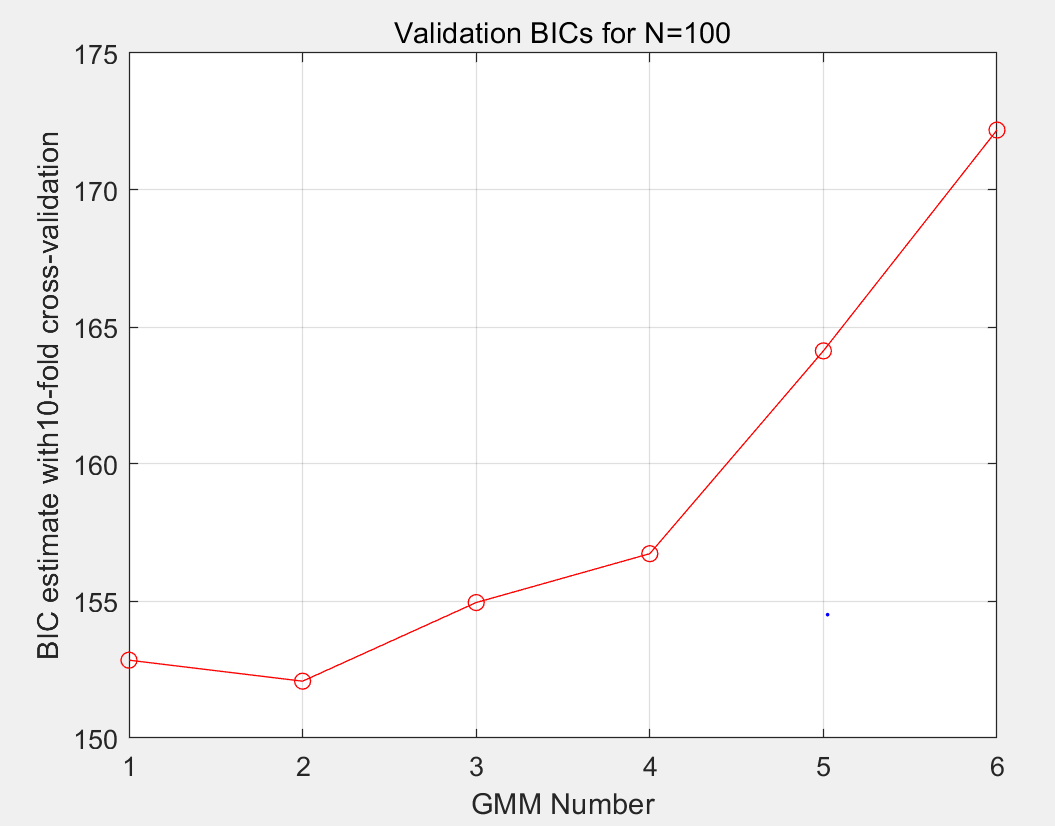
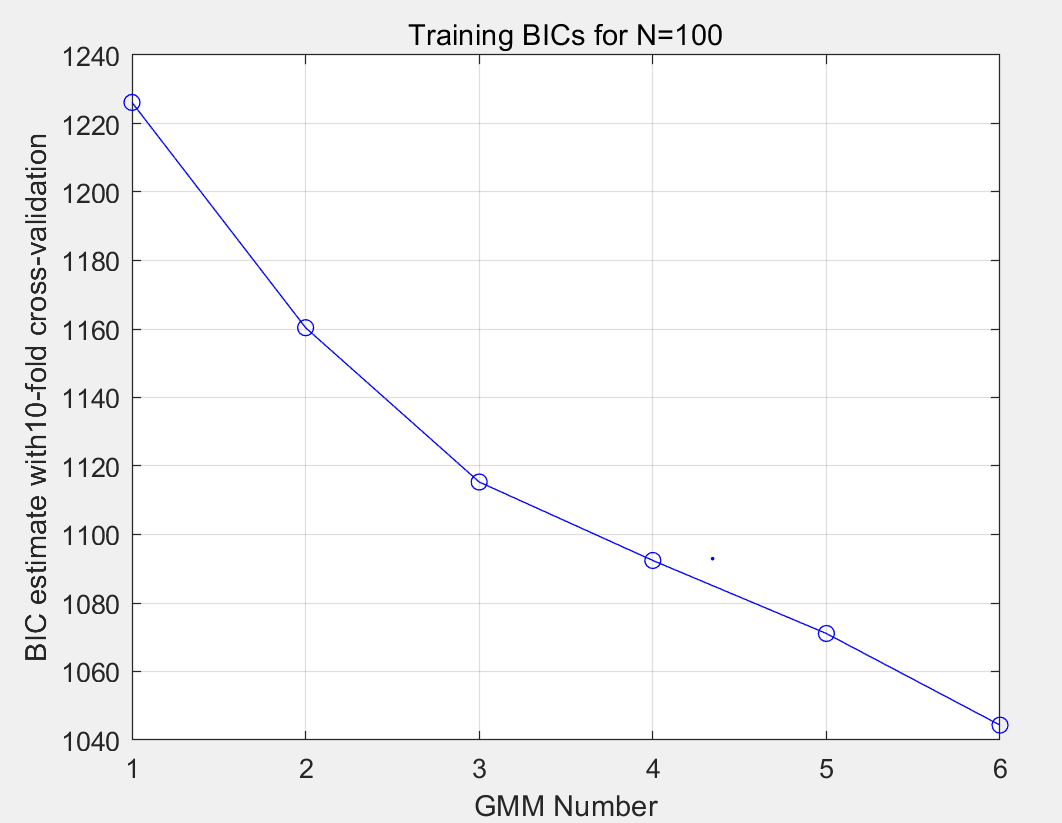
N=100



*Figure 14 4class for N=100*

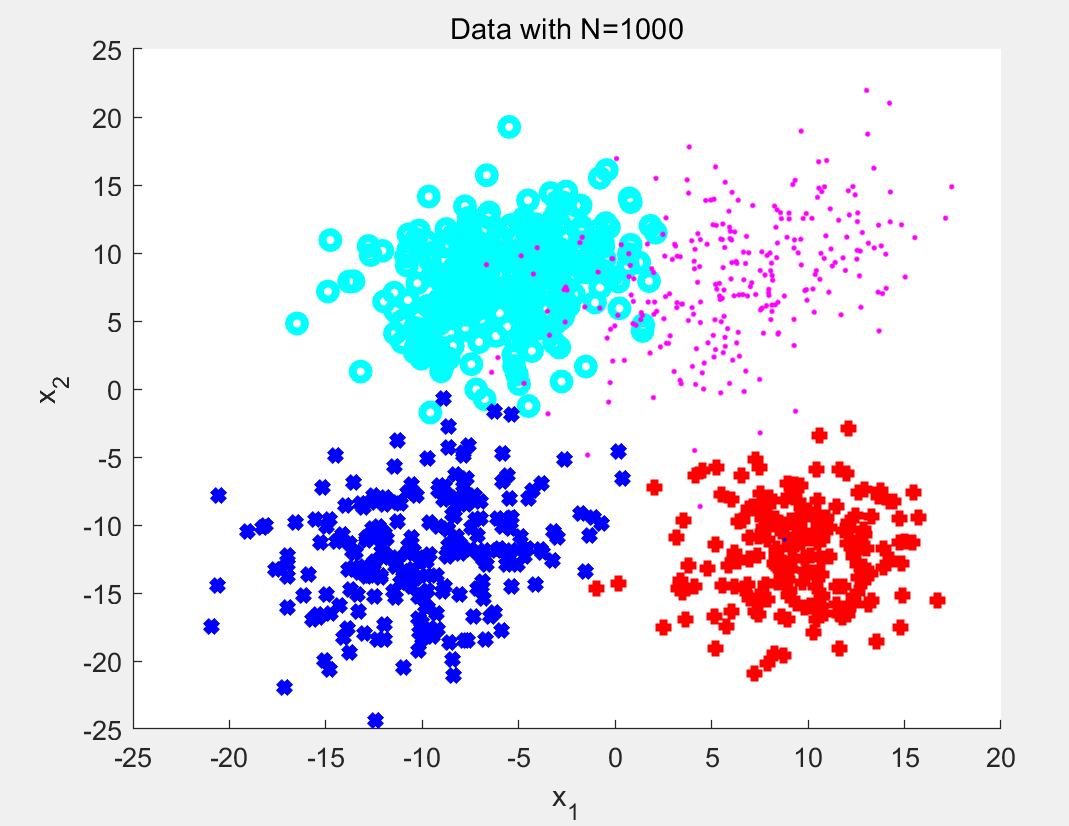


*Figure 15 loglikelihood for N=100*

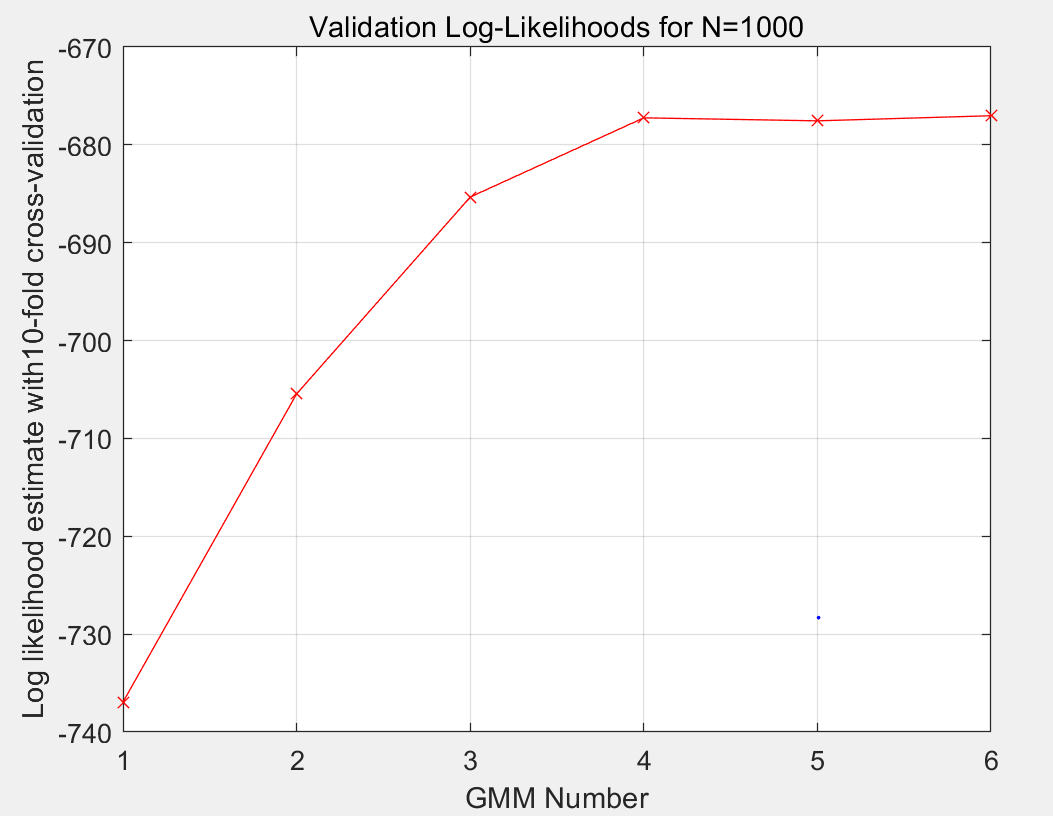
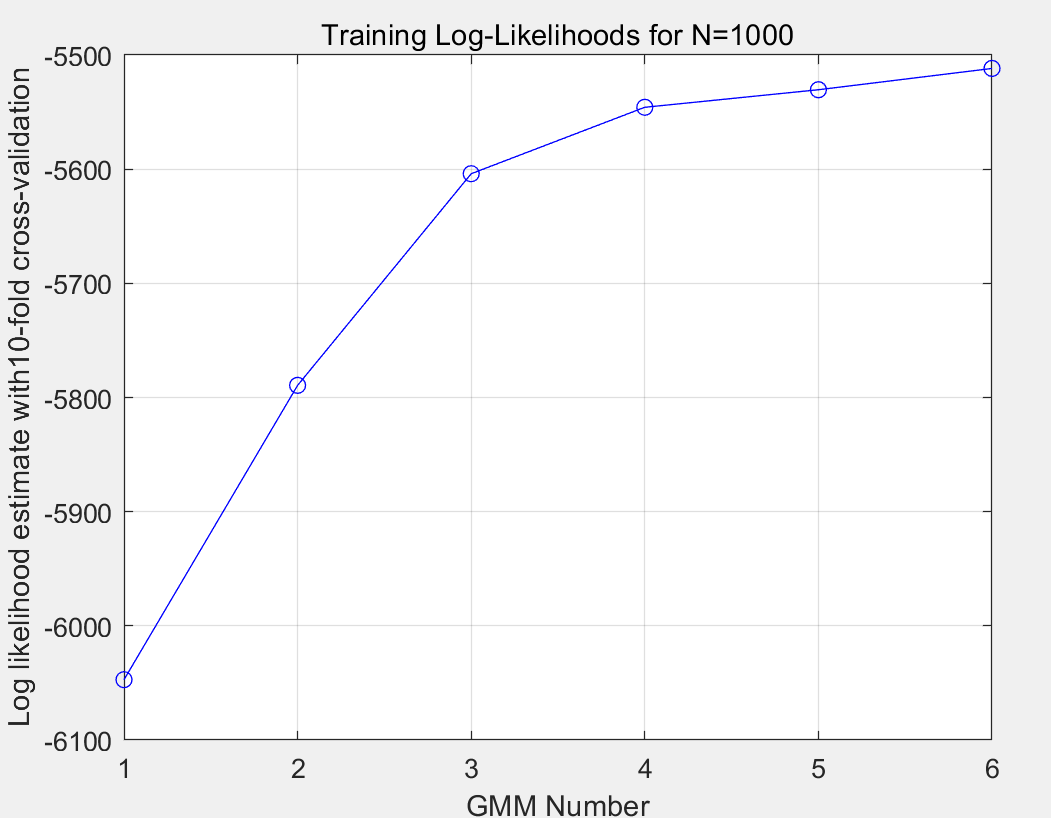


*Figure 16 BIC for N=100*

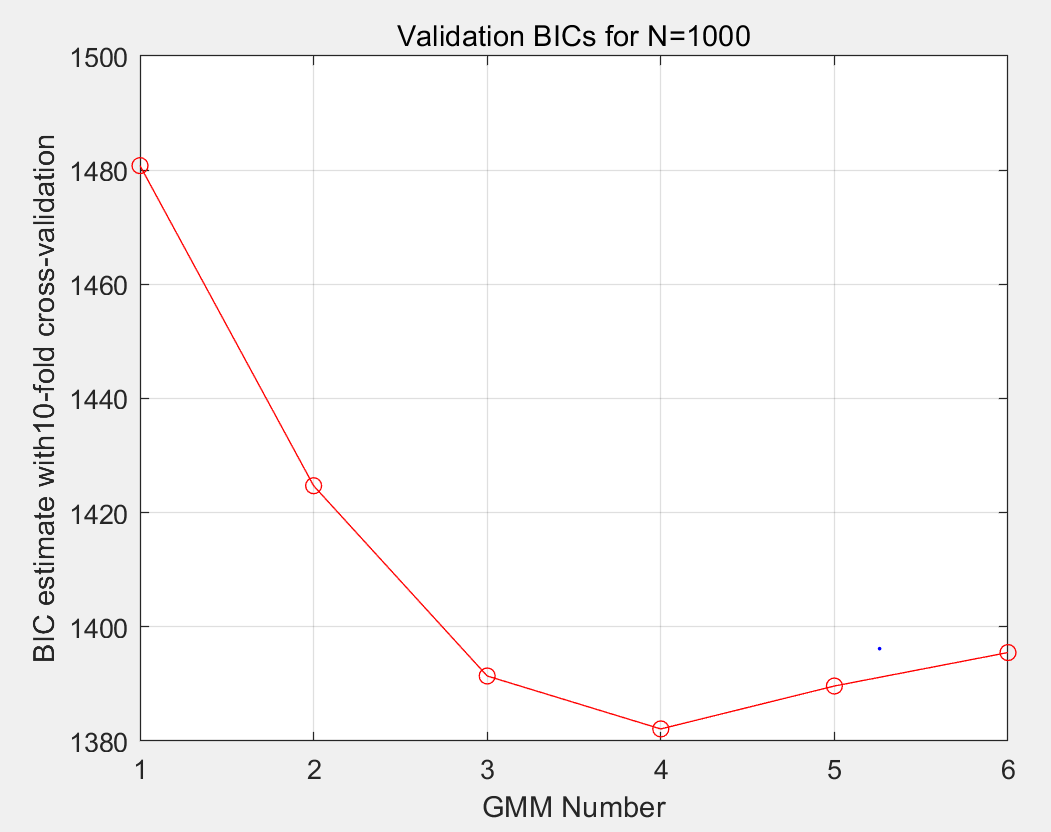
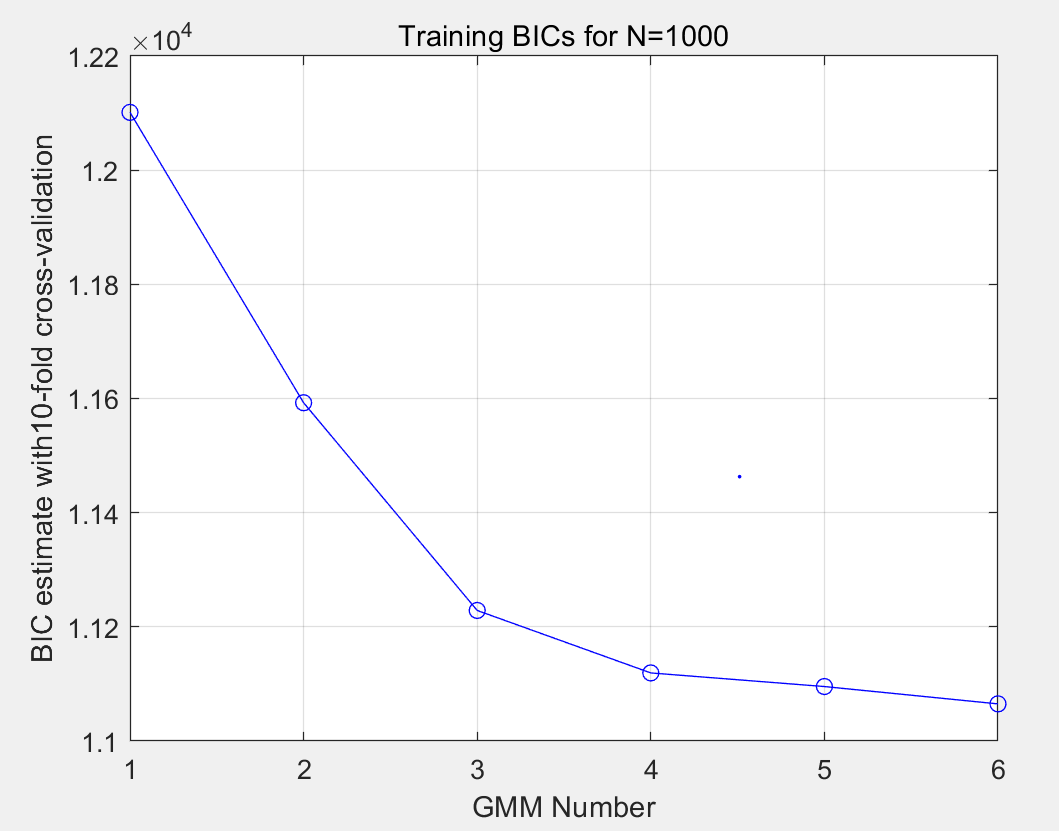
N=1000



*Figure 17 4class for N=1000*

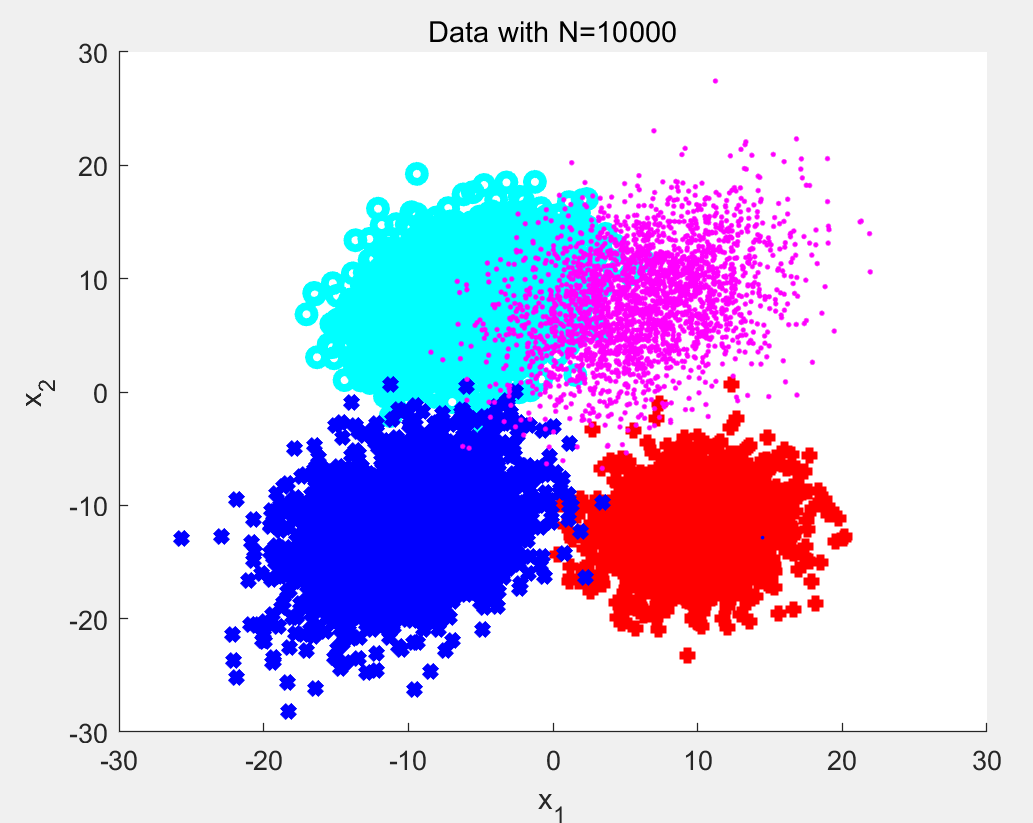


*Figure 18 loglikelihood for N=1000*

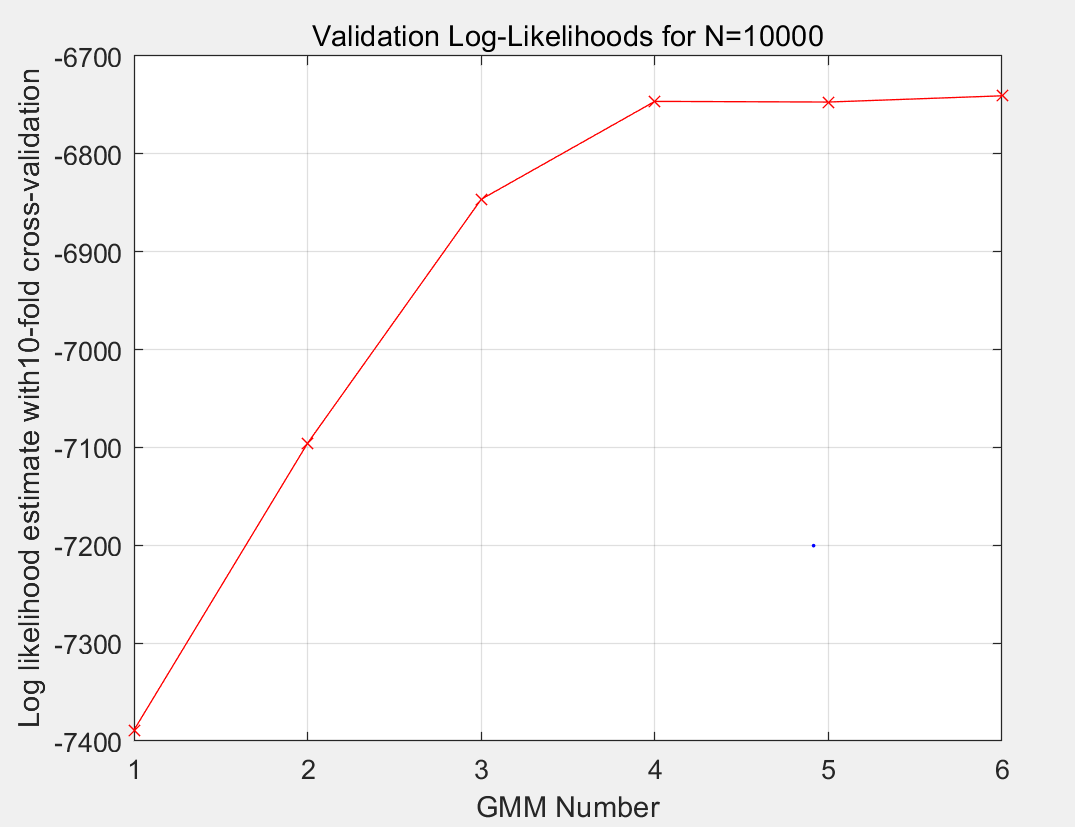
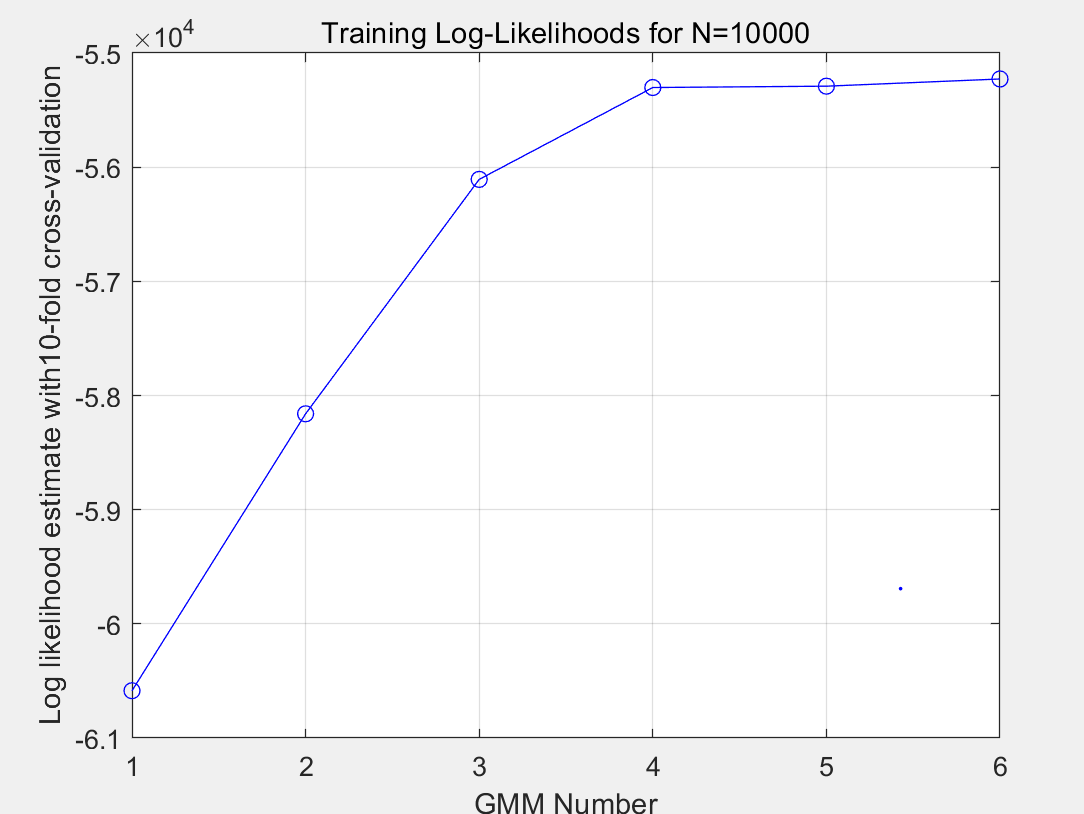


*Figure 19 BIC for N=1000*

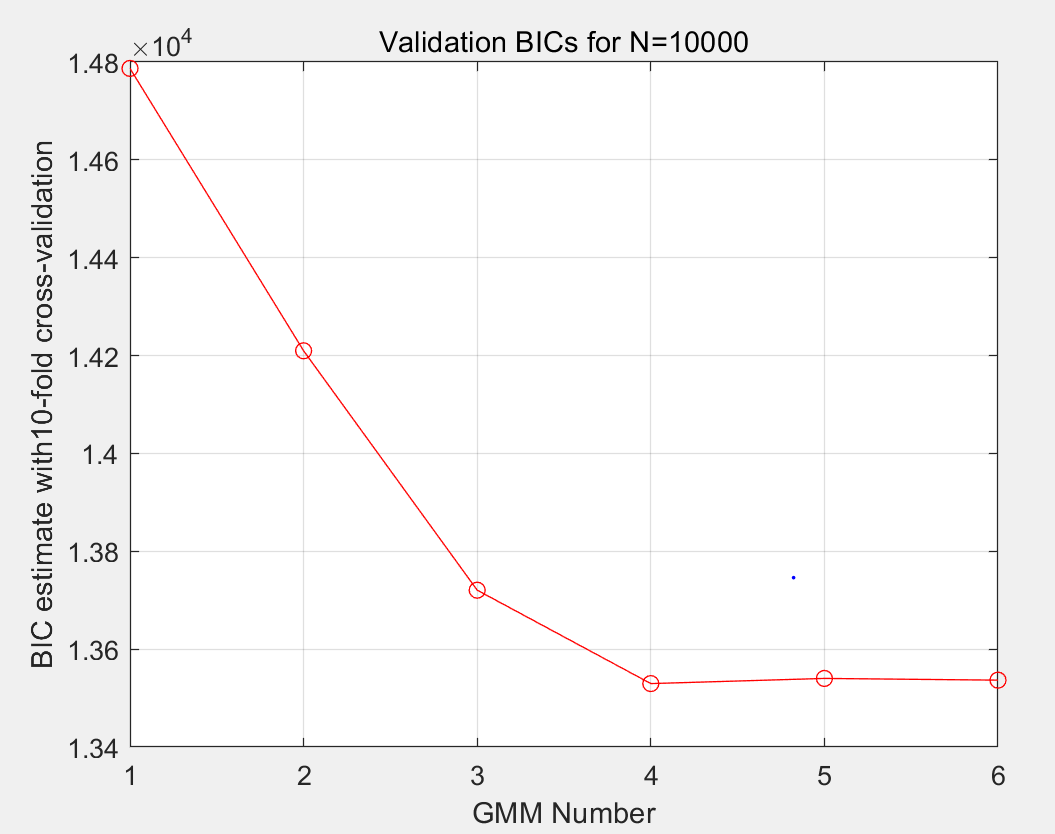
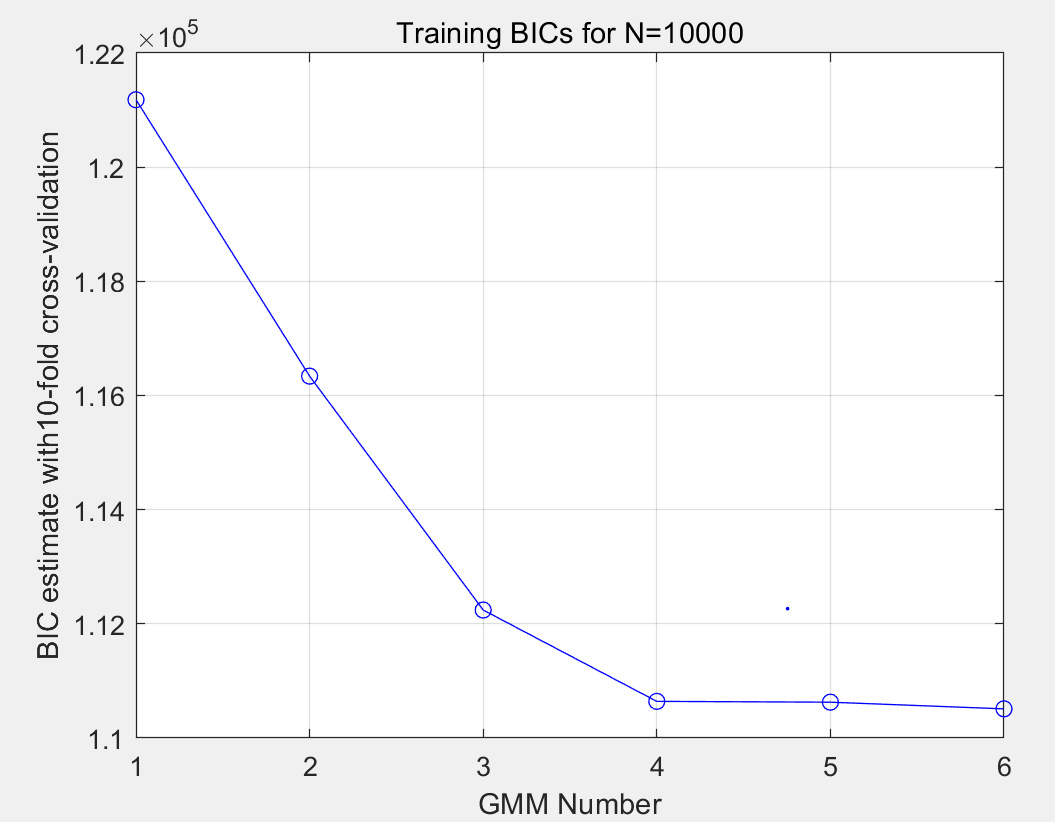
N=10000



*Figure 20 4class for N=10000*



*Figure 21 loglikelihood for N=10000*



*Figure 22 BIC for N=10000*

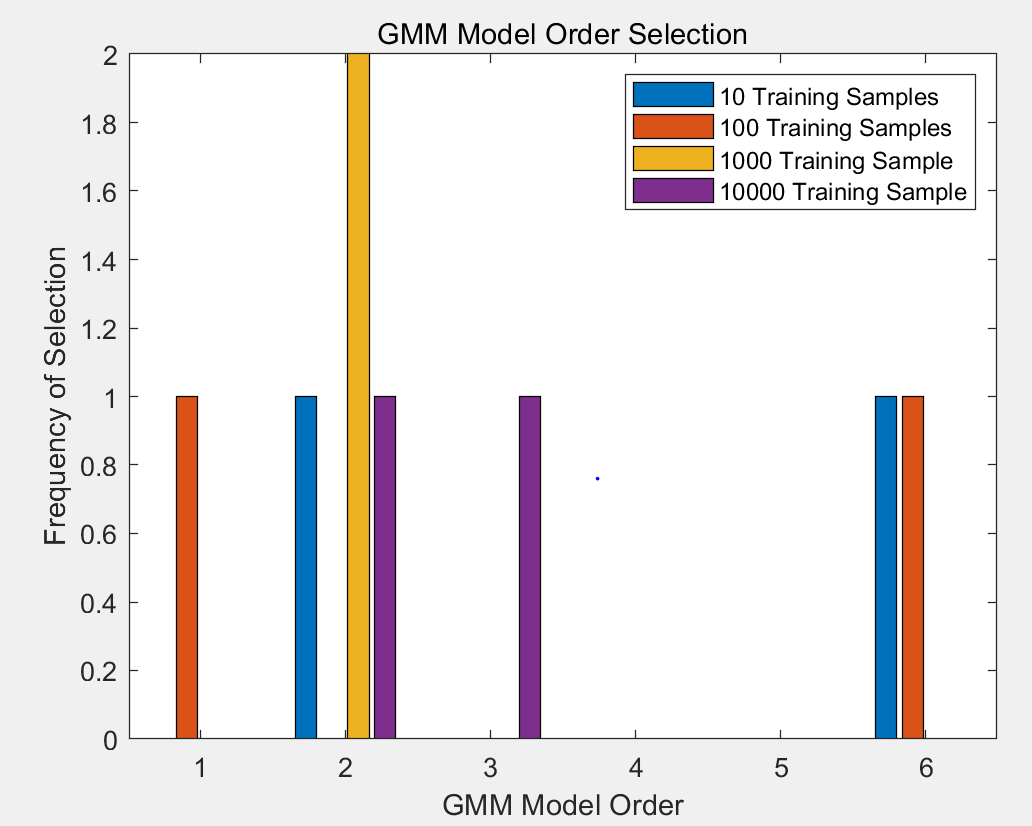
Conclusion:

The log likelihood and BIC plot at number of 1000 and 10000(which may be more accurate) indicate number of components = 4 is a good model for these data. We can see from figures above,when the order equals 4, the dots rise sharply and flatten in log likelihood plot,so the order of 4 is the best.Also, from the BIC plot, we can easily get the same conclusion.

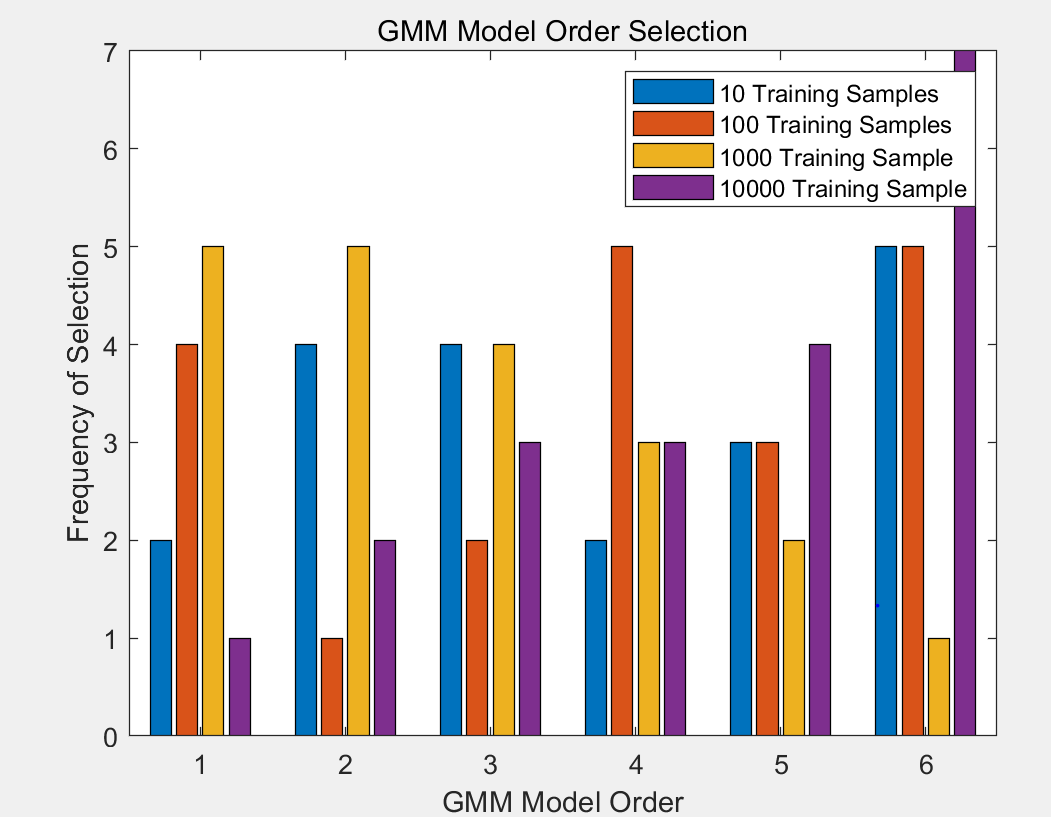
Since after adjusting the original distribution several times,I could not get the algorithm converging in all cases,the algorithm has be limited for each combination of parameters. Although this does not provide a solution for the convergence issue, I still can get some data to show the performance of the algorithm.

In figure 23,the plot shows the results of running the algorithm

for the first experiment and keeping track of when each component number was selected as the winner and in figure 24,the plot shows the last experiment.



*Figure 23 1st experiment*



*Figure 24 30th experiment*

when only 10 data points were used to form the training data set, most

of the iterations ended with lower-valued component numbers being selected (only 2 and 6). This makes sense since the 10 points would be very spread out and it would be difficult to form as many as 6 groups based on just those 10 points. However,with 100,1000 and 10000 sample data sets, the distribution was also lower-valued component numbers and scattered in different components, this may because that the less experiment may not be very accurate.On the other hand, it may because my origin distributions are very scattered.

In figure 24,we can find that with more points at disposal to form the training set and more experiment, the algorithm can pick the correct component number more often.

**Appendix:**

**Q1:**

%%=======================Question1=========================%%

% Dandan lin/001093902

% Code help and example from Prof.Deniz

clear all;

close all;

%Switches to bypass parts 1 and 2 for debugging

dimensions=3; %Dimension of data

numLabels=4;

Lx={'L0', 'L1','L2','L3'};

% For min-Perror design, use 0-1 loss

lossMatrix = ones(numLabels,numLabels)-eye(numLabels);

muScale=2;

SigmaScale=0.2;

%Define data

D.d100.N=100;

D.d200.N=200;

D.d500.N=500;

D.d1k.N=1e3;

D.d2k.N=2e3;

D.d5k.N=5e3;

D.d100k.N=100e3;

dTypes=fieldnames(D);

%Define Statistics

p=ones(1,numLabels)/numLabels; %Prior

%Label data stats

mu.L0=muScale\*[1 1 0]';

RandSig=SigmaScale\*rand(dimensions,dimensions);

Sigma.L0(:,:,1)=RandSig\*RandSig'+eye(dimensions);

mu.L1=muScale\*[0 1 0]';

RandSig=SigmaScale\*rand(dimensions,dimensions);

Sigma.L1(:,:,1)=RandSig\*RandSig'+eye(dimensions);

mu.L2=muScale\*[0 0 1]';

RandSig=SigmaScale\*rand(dimensions,dimensions);

Sigma.L2(:,:,1)=RandSig\*RandSig'+eye(dimensions);

mu.L3=muScale\*[1 0 1]';

RandSig=SigmaScale\*rand(dimensions,dimensions);

Sigma.L3(:,:,1)=RandSig\*RandSig'+eye(dimensions);

%%Generate Data%%

for ind=1:length(dTypes)

D.(dTypes{ind}).x=zeros(dimensions,D.(dTypes{ind}).N); %Initialize Data

[D.(dTypes{ind}).x,D.(dTypes{ind}).labels,...

D.(dTypes{ind}).N\_l,D.(dTypes{ind}).p\_hat]=...

genData(D.(dTypes{ind}).N,p,mu,Sigma,Lx,dimensions);

end

%Plot Training Data

figure;

for ind=1:length(dTypes)-1

subplot(3,2,ind);

plotData(D.(dTypes{ind}).x,D.(dTypes{ind}).labels,Lx);

legend 'show';

title([dTypes{ind}]);

end

%Plot Validation Data

figure;

plotData(D.(dTypes{ind}).x,D.(dTypes{ind}).labels,Lx);

legend 'show';

title([dTypes{end}]);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Determine Theoretically Optimal Classifier

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for ind=1:length(dTypes)

[D.(dTypes{ind}).opt.PFE, D.(dTypes{ind}).opt.decisions]=...

optClass(lossMatrix,D.(dTypes{ind}).x,mu,Sigma,...

p,D.(dTypes{ind}).labels,Lx);

opPFE(ind)=D.(dTypes{ind}).opt.PFE;

fprintf('Optimal pFE, N=%1.0f: Error=%1.2f%%\n',...

D.(dTypes{ind}).N,100\*D.(dTypes{ind}).opt.PFE);

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Train and Validate Data

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

numPerc=15; %Max number of perceptrons to attempt to train

k=10; %number of folds for kfold validation

for ind=1:length(dTypes)-1

%kfold validation is in this function

[D.(dTypes{ind}).net,D.(dTypes{ind}).minPFE,...

D.(dTypes{ind}).optM,valData.(dTypes{ind}).stats]=...

kfoldMLP\_NN(numPerc,k,D.(dTypes{ind}).x,...

D.(dTypes{ind}).labels,numLabels);

%Produce validation data from test dataset

valData.(dTypes{ind}).yVal=D.(dTypes{ind}).net(D.d100k.x);

[~,valData.(dTypes{ind}).decisions]=max(valData.(dTypes{ind}).yVal);

valData.(dTypes{ind}).decisions=valData.(dTypes{ind}).decisions-1;

%Probability of Error is wrong decisions/num data points

valData.(dTypes{ind}).pFE=...

sum(valData.(dTypes{ind}).decisions~=D.d100k.labels)/D.d100k.N;

outpFE(ind,1)=D.(dTypes{ind}).N;

outpFE(ind,2)=valData.(dTypes{ind}).pFE;

outpFE(ind,3)=D.(dTypes{ind}).optM;

fprintf('NN pFE, N=%1.0f: Error=%1.2f%%\n',...

D.(dTypes{ind}).N,100\*valData.(dTypes{ind}).pFE);

end

%This code was used to plot the results from the data generated in the main

%function

%Extract cross validation results from structure

for ind=1:length(dTypes)-1

[~,select]=min(valData.(dTypes{ind}).stats.mPFE);

M(ind)=(valData.(dTypes{ind}).stats.M(select));

N(ind)=D.(dTypes{ind}).N;

end

%Plot number of perceptrons vs. pFE for the cross validation runs

for ind=1:length(dTypes)-1

figure;

stem(valData.(dTypes{ind}).stats.M,valData.(dTypes{ind}).stats.mPFE);

xlabel('Number of Perceptrons');

ylabel('pFE');

title(['Probability of Error vs. Number of Perceptrons for ' dTypes{ind}]);

end

%Number of perceptrons vs. size of training dataset

figure,semilogx(N(1:end-1),M(1:end-1),'o','LineWidth',2)

grid on;

xlabel('Number of Data Points')

ylabel('Optimal Number of Perceptrons')

ylim([0 10]);

xlim([50 10^4]);

title('Optimal Number of Perceptrons vs. Number of Data Points');

%Prob. of Error vs. size of training data set

figure,semilogx(outpFE(1:end-1,1),outpFE(1:end-1,2),'o','LineWidth',2)

xlim([90 10^4]);

hold all;semilogx(xlim,[opPFE(end) opPFE(end)],'r--','LineWidth',2)

legend('NN pFE','Optimal pFE')

grid on

xlabel('Number of Data Points')

ylabel('pFE')

title('Probability of Error vs. Data Points in Training Data');

Function:

function [indexMAP, pEminERM] = classifyMAP(data, classIndex, mu, sigma, nSamples, prior)

% Expected Risk Minimization Classifier%

discriminantScoreERM = log(evalGaussian(data',mu(2,:)',sigma(:,:,2)))- log(evalGaussian(data',mu(1,:)',sigma(:,:,1)));

% MAP classifier (is a special case of ERM corresponding to 0-1 loss)

lambdaMAP = [0 1;1 0]; % 0-1 loss values yield MAP decision rule

gammaMAP = (lambdaMAP(2, 1)-lambdaMAP(1,1))/(lambdaMAP(1,2)-lambdaMAP(2,2)) \* prior(1)/prior(2); % threshold for MAP

decisionMAP = (discriminantScoreERM >= log(gammaMAP));

ind00MAP = find(decisionMAP==0 & classIndex'==1); p00MAP = length(ind00MAP)/ sum(classIndex==1); % probability of true negative

ind10MAP = find(decisionMAP==1 & classIndex'==1); p10MAP = length(ind10MAP)/ sum(classIndex==1); % probability of false positive

ind01MAP = find(decisionMAP==0 & classIndex'==2); p01MAP = length(ind01MAP)/ sum(classIndex==2); % probability of false negative

ind11MAP = find(decisionMAP==1 & classIndex'==2); p11MAP = length(ind11MAP)/ sum(classIndex==2); % probability of true positive

pEminERM = [p10MAP,p01MAP]\*[sum(classIndex==1),sum(classIndex==2)]'/nSamples; % probability of error for MAP classifier, empirically estimated

indexMAP{1} = ind00MAP;

indexMAP{2} = ind10MAP;

indexMAP{3} = ind01MAP;

indexMAP{4} = ind11MAP;

fprintf('MAP error number: %2f \n',length(ind10MAP)+length(ind01MAP));

fprintf('MAP probability of error %2f \n',pEminERM);

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 [x,labels,N\_l,p\_hat]= genData(N,p,mu,Sigma,Lx,d)

%Generates data and labels for random variable x from multiple gaussian

%distributions

numD = length(Lx);

cum\_p = [0,cumsum(p)];

u = rand(1,N);

x = zeros(d,N);

labels = zeros(1,N);

for ind=1:numD

pts = find(cum\_p(ind)<u & u<=cum\_p(ind+1));

N\_l(ind)=length(pts);

x(:,pts) = mvnrnd(mu.(Lx{ind}),Sigma.(Lx{ind}),N\_l(ind))';

labels(pts)=ind-1;

p\_hat(ind)=N\_l(ind)/N;

end

End

function [outputNet,outputPFE, optM,stats]=kfoldMLP\_NN(numPerc,k,x,labels,numLabels)

%Assumes data is evenly divisible by partition choice which it should be

N=length(x);

numValIters=10;

%Create output matrices from labels

y=zeros(numLabels,length(x));

for ind=1:numLabels

y(ind,:)=(labels==ind-1);

end

%Setup cross validation on training data

partSize=N/k;

partInd=[1:partSize:N length(x)];

%Perform cross validation to select number of perceptrons

for M=1:numPerc

for ind=1:k

index.val=partInd(ind):partInd(ind+1);

index.train=setdiff(1:N,index.val);

%Create object with M perceptrons in hidden layer

net=patternnet(M);

% net.layers{1}.transferFcn = 'softplus';%didn't work

%Train using training data

net=train(net,x(:,index.train),y(:,index.train));

%Validate with remaining data

yVal=net(x(:,index.val));

[~,labelVal]=max(yVal);

labelVal=labelVal-1;

pFE(ind)=sum(labelVal~=labels(index.val))/partSize;

end

%Determine average probability of error for a number of perceptrons

avgPFE(M)=mean(pFE);

stats.M=1:M;

stats.mPFE=avgPFE;

end

%Determine optimal number of perceptrons

[~,optM]=min(avgPFE);

%Train one final time on all the data

for ind=1:numValIters

netName(ind)={['net' num2str(ind)]};

finalnet.(netName{ind})=patternnet(optM);

% finalnet.layers{1}.transferFcn = 'softplus';%Set to RELU

finalnet.(netName{ind})=train(net,x,y);

yVal=finalnet.(netName{ind})(x);

[~,labelVal]=max(yVal);

labelVal=labelVal-1;

pFEFinal(ind)=sum(labelVal~=labels)/length(x);

end

[minPFE,outInd]=min(pFEFinal);

stats.finalPFE=pFEFinal;

outputPFE=minPFE;

outputNet=finalnet.(netName{outInd});

End

function [minPFE,decisions]=optClass(lossMatrix,x,mu,Sigma,p,labels,Lx)

% Determine optimal probability of error

symbols='ox+\*v';

numLabels=length(Lx);

N=length(x);

for ind = 1:numLabels

pxgivenl(ind,:) =...

evalGaussian(x,mu.(Lx{ind}),Sigma.(Lx{ind})); % Evaluate p(x|L=l)

end

px = p\*pxgivenl; % Total probability theorem

classPosteriors = pxgivenl.\*repmat(p',1,N)./repmat(px,numLabels,1); %P(L=l|x)

% Expected Risk for each label (rows) for each sample (columns)

expectedRisks = lossMatrix\*classPosteriors;

% Minimum expected risk decision with 0-1 loss is the same as MAP

[~,decisions] = min(expectedRisks,[],1);

decisions=decisions-1; %Adjust to account for L0 label

fDecision\_ind=(decisions~=labels);%Incorrect classificiation vector

minPFE=sum(fDecision\_ind)/N;

%Plot Decisions with Incorrect Results

figure;

for ind=1:numLabels

class\_ind=decisions==ind-1;

plot3(x(1,class\_ind & ~fDecision\_ind),...

x(2,class\_ind & ~fDecision\_ind),...

x(3,class\_ind & ~fDecision\_ind),...

symbols(ind),'Color',[0.39 0.83 0.07],'DisplayName',...

['Class ' num2str(ind) ' Correct Classification']);

hold on;

plot3(x(1,class\_ind & fDecision\_ind),...

x(2,class\_ind & fDecision\_ind),...

x(3,class\_ind & fDecision\_ind),...

['r' symbols(ind)],'DisplayName',...

['Class ' num2str(ind) ' Incorrect Classification']);

hold on;

end

xlabel('x1');

ylabel('x2');

grid on;

title('X Vector with Incorrect Classifications');

legend 'show';

if 0

%Plot Decisions with Incorrect Decisions

figure;

for ind2=1:numLabels

subplot(3,2,ind2);

for ind=1:numLabels

class\_ind=decisions==ind-1;

plot3(x(1,class\_ind),x(2,class\_ind),x(3,class\_ind),...

'.','DisplayName',['Class ' num2str(ind)]);

hold on;

end

plot3(x(1,fDecision\_ind & labels==ind2),...

x(2,fDecision\_ind & labels==ind2),...

x(3,fDecision\_ind & labels==ind2),...

'kx','DisplayName','Incorrectly Classified','LineWidth',2);

ylabel('x2');

grid on;

title(['X Vector with Incorrect Decisions for Class '

num2str(ind2)]);

if ind2==1

legend 'show';

elseif ind2==4

xlabel('x1');

end

end

end

End

function plotData(x,labels,Lx)

%Plots data

for ind=1:length(Lx)

pindex=labels==ind-1;

plot3(x(1,pindex),x(2,pindex),x(3,pindex),'.','DisplayName',Lx{ind});

hold all;

end

grid on;

xlabel('x1');

ylabel('x2');

zlabel('x3');

end

**Q2:**

%%=======================Question2=========================%%

% examples by Prof.Deniz

close all; clear; clc;

N=10;

d=2;

alpha\_true=[0.2,0.3,0.24,0.26];

mu\_true(:,1) = [10;-12];

mu\_true(:,2) = [-6;8];

mu\_true(:,3) = [-10;-12];

mu\_true(:,4) = [6;8];

Sigma\_true(:,:,1) = [10 1;1 10];

Sigma\_true(:,:,2) = [11 3;3 11];

Sigma\_true(:,:,3) = [17 5;5 17];

Sigma\_true(:,:,4) = [21 7;7 21]; % Number of samples

N=[10,100,1000,10000]; % ensure the program is not stuck

countN = 0;

num\_GMM\_picks = zeros(length(N),6);

num\_GMM\_cmp = zeros(length(N),6);

for i=1:length(N)

countN = countN+1

% Create appropriate number of data points from each distribution

[x,label]=generate\_samples(N(i),mu\_true,Sigma\_true,alpha\_true);

% plot

figure(i);

scatter(x(1,label==1),x(2,label==1),'r','+','LineWidth',3);

hold on

scatter(x(1,label==2),x(2,label==2),'c','o','LineWidth',3);

hold on

scatter(x(1,label==3),x(2,label==3),'b','x','LineWidth',3);

hold on

scatter(x(1,label==4),x(2,label==4),'m','.','LineWidth',3);

title(strcat('Data with N=',num2str(N(i))));

xlabel('x\_1'),ylabel('x\_2')

saveas(gcf,['./Q2figs/',int2str(i),'.jpg']);

GMM\_pick=cross\_val(x);

num\_GMM\_picks(i,GMM\_pick)=num\_GMM\_picks(i,GMM\_pick)+1;

%Tolerance for EM stopping criterion

delta = 1e-4;

%Regularization parameter for covariance estimates

regWeight = 1e-10;

%K-Fold Cross Validation

K = 10;

%To determine dimensionality of samples and number of GMM components

[d,MM] = size(mu\_true);

%Divide the data set into 10 approximately-equal-sized partitions

dummy = ceil(linspace(0,N(i),K+1));

for k = 1:K

indPartitionLimits(k,:) = [dummy(k)+1,dummy(k+1)];

end

%Allocate space

loglikelihoodtrain = zeros(K,6); loglikelihoodvalidate = zeros(K,6);

Averagelltrain = zeros(1,6); Averagellvalidate = zeros(1,6);

countM = 0;

%Try all 6 mixture options

for M = 1:6

countM = countM+1

countk = 0;

%10-fold cross validation

for k = 1:K

countk = countk+1

indValidate = [indPartitionLimits(k,1):indPartitionLimits(k,2)];

%Using folk k as validation set

x1Validate = x(1,indValidate);

x2Validate = x(2,indValidate);

if k == 1

indTrain = [indPartitionLimits(k,2)+1:N(i)];

elseif k == K

indTrain = [1:indPartitionLimits(k,1)-1];

else

indTrain = [1:indPartitionLimits(k-1,2),indPartitionLimits(k+1,2):N(i)];

end

%Using all other folds as training set

x1Train = x(1,indTrain);

x2Train = x(2,indTrain);

xTrain = [x1Train; x2Train];

xValidate = [x1Validate; x2Validate];

Ntrain = length(indTrain); Nvalidate = length(indValidate);

%Train model parameters (EM)

%Initialize the GMM to randomly selected samples

alpha = ones(1,M)/M;

shuffledIndices = randperm(Ntrain);

%Pick M random samples as initial mean estimates (this led

%to good initial estimates (better log likelihoods))

mu = xTrain(:,shuffledIndices(1:M));

%Assign each sample to the nearest mean (better initialization)

[~,assignedCentroidLabels] = min(pdist2(mu',xTrain'),[],1);

%Use sample covariances of initial assignments as initial covariance estimates

for m = 1:M

Sigma(:,:,m) = cov(xTrain(:,find(assignedCentroidLabels==m))') + regWeight\*eye(d,d);

end

t = 0;

%Not converged at the beginning

Converged = 0;

while ~Converged

for l = 1:M

temp(l,:) = repmat(alpha(l),1,Ntrain).\*evalGaussian(xTrain,mu(:,l),Sigma(:,:,l));

end

plgivenx = temp./sum(temp,1);

clear temp

alphaNew = mean(plgivenx,2);

w = plgivenx./repmat(sum(plgivenx,2),1,Ntrain);

muNew = xTrain\*w';

for l = 1:M

v = xTrain-repmat(muNew(:,l),1,Ntrain);

u = repmat(w(l,:),d,1).\*v;

%Adding a small regularization term

SigmaNew(:,:,l) = u\*v' + regWeight\*eye(d,d);

end

Dalpha = sum(abs(alphaNew-alpha));

Dmu = sum(sum(abs(muNew-mu)));

DSigma = sum(sum(abs(abs(SigmaNew-Sigma))));

%Check if converged

Converged = ((Dalpha+Dmu+DSigma)<delta);

alpha = alphaNew; mu = muNew; Sigma = SigmaNew;

t = t+1;

end

%Validation

loglikelihoodtrain(k,M) = sum(log(evalGMM(xTrain,alpha,mu,Sigma)));

loglikelihoodvalidate(k,M) = sum(log(evalGMM(xValidate,alpha,mu,Sigma)));

end

%Average Performance Variables

Averagelltrain(1,M) = mean(loglikelihoodtrain(:,M));

BICtrain(1,M) = -2\*Averagelltrain(1,M)+M\*log(N(i));

Averagellvalidate(1,M) = mean(loglikelihoodvalidate(:,M));

%Sometimes the log likelihoods for N=10 are zero, leading to

%negative infinity results. I assume that this is instead the

%lowest log likelihood value instead (so it is possible to graph).

if isinf(Averagellvalidate(1,M))

Averagellvalidate(1,M) = (min(Averagellvalidate(find(isfinite(Averagellvalidate)))));

end

BICvalidate(1,M) = -2\*Averagellvalidate(1,M)+M\*log(N(i));

%Recording values

TotBICValidate(i,M) = BICvalidate(1,M);

TotBICTrain(i,M) = BICtrain(1,M);

TotAvgllValidate(i,M) = Averagellvalidate(1,M);

TotAvgllTrain(i,M) = Averagelltrain(1,M);

end

%Recording Best Outcomes

[LowestBIC orderB] = min(BICvalidate)

[Lowestll orderl] = max(Averagellvalidate)

% training log-likelihood

figure(i+4), clf,

plot(Averagelltrain,'ob');

hold on;

plot(Averagelltrain,'-b');

xlabel('GMM Number'); ylabel(strcat('Log likelihood estimate with ',num2str(K),'-fold cross-validation'));

title(strcat('Training Log-Likelihoods for N=',num2str(N(i))));

grid on

xticks(1:1:6)

saveas(gcf,['./Q2figs/',int2str(i+4),'.jpg']);

% validation log-likelihood

figure(i+8), clf,

plot(Averagellvalidate,'rx');

hold on;

plot(Averagellvalidate,'r-');

xlabel('GMM Number'); ylabel(strcat('Log likelihood estimate with ',num2str(K),'-fold cross-validation'));

title(strcat('Validation Log-Likelihoods for N=',num2str(N(i))));

grid on

xticks(1:1:6)

saveas(gcf,['./Q2figs/',int2str(i+8),'.jpg']);

% training BIC

figure(i+12), clf,

plot(BICtrain,'ob');

hold on;

plot(BICtrain,'b');

xlabel('GMM Number'); ylabel(strcat('BIC estimate with ',num2str(K),'-fold cross-validation'));

title(strcat('Training BICs for N=',num2str(N(i))));

grid on

xticks(1:1:6)

saveas(gcf,['./Q2figs/',int2str(i+12),'.jpg']);

% validation BIC

figure(i+16), clf,

plot(BICvalidate,'ro');

hold on;

plot(BICvalidate,'r');

xlabel('GMM Number'); ylabel(strcat('BIC estimate with ',num2str(K),'-fold cross-validation'));

title(strcat('Validation BICs for N=',num2str(N(i))))

grid on

xticks(1:1:6)

saveas(gcf,['./Q2figs/',int2str(i+16),'.jpg']);

%Saving values

BICorder(i) = orderB;

BIClow(i) = LowestBIC;

lorder(i) = orderl;

lllow(i) = Lowestll;

end

% multi experiments

for a=1:30

for i=1:length(N)

x,label=generate\_samples(N(i),mu\_true,Sigma\_true,alpha\_true);

GMM\_pick=cross\_val(x);

num\_GMM\_picks(i,GMM\_pick)=num\_GMM\_picks(i,GMM\_pick)+1;

end

if ~isequal(num\_GMM\_cmp, num\_GMM\_picks)

figure,

bar(num\_GMM\_picks');

legend('10 Training Samples','100 Training Samples', ...

'1000 Training Sample','10000 Training Sample');

title('GMM Model Order Selection');

xlabel('GMM Model Order');ylabel('Frequency of Selection');

saveas(gcf,['./Q2figs/4-',int2str(a),'.jpg']);

num\_GMM\_cmp=num\_GMM\_picks;

end

end

%%====================Question 2 Functions====================%%

% Functions credit to Prof.Deniz

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) = randGaussian(length(ind),mu(:,m),Sigma(:,:,m));

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 column 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 best\_GMM=cross\_val(x)

B=10;M=6;%repetitionsperdataset;maxGMMconsidered

perf\_array=zeros(B,M);%savespaceforperformanceevaluation

%Testeachdataset10times

for b=1:B

%add noise

set\_size=500;

train\_index=randi([1,length(x)],[1,set\_size]);

train\_set=x(:,train\_index)+(1e-3)\*randn(2,set\_size);

val\_index=randi([1,length(x)],[1,set\_size]);

val\_set=x(:,val\_index)+(1e-3)\*randn(2,set\_size);

for m=1:M

GMModel=fitgmdist(train\_set',M,'RegularizationValue',1e-10);

alpha=GMModel.ComponentProportion;

mu=(GMModel.mu)';

sigma=GMModel.Sigma;

perf\_array(b,m)=sum(log(evalGMM(val\_set,alpha,mu,sigma)));

end

End

avg\_perf=sum(perf\_array)/B;

best\_GMM=find(avg\_perf==max(avg\_perf),1);

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [x,label]=generate\_samples(N,mu\_true,Sigma\_true,alpha\_true)

% Create appropriate number of data points from each distribution

x=zeros(2,N);

label=zeros(1,N);

for j=1:N

r=rand(1);

if r <= alpha\_true(1)

label(j)=1;

elseif (alpha\_true(1)<r)&&(r<=sum(alpha\_true(1:2)))

label(j)=2;

elseif (sum(alpha\_true(1:2))<r)&&(r<=sum(alpha\_true(1:3)))

label(j)=3;

else

label(j)=4;

end

end

Nc=[sum(label==1),sum(label==2),sum(label==3),sum(label==4)];

% Generate data

x(:,label==1)=randGaussian(Nc(1),mu\_true(:,1),Sigma\_true(:,:,1));

x(:,label==2)=randGaussian(Nc(2),mu\_true(:,2),Sigma\_true(:,:,2));

x(:,label==3)=randGaussian(Nc(3),mu\_true(:,3),Sigma\_true(:,:,3));

x(:,label==4)=randGaussian(Nc(4),mu\_true(:,4),Sigma\_true(:,:,4));

end