Report for the Assignment 3 of Machine Learning and Pattern Recognition

Question 1 :

For this problem many multilayer perceptron (MLP) were used to approximate class label posteriors. Minimum average cross-entropy loss was used to train the MLP and the trained models were then used to approximate a MAP classification rule to achieve minimum probability of error on a validation dataset.
For this exercise 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.
$P(L=l)=0.2, for\ l=[0,1,2,3]$

$\boldsymbol{m}0$=[2.6 2.6 0],$\boldsymbol{C}0$=[1.00 0.01 0.01 , 0.01 1.03 0.02 , 0.01 0.06 1.03]

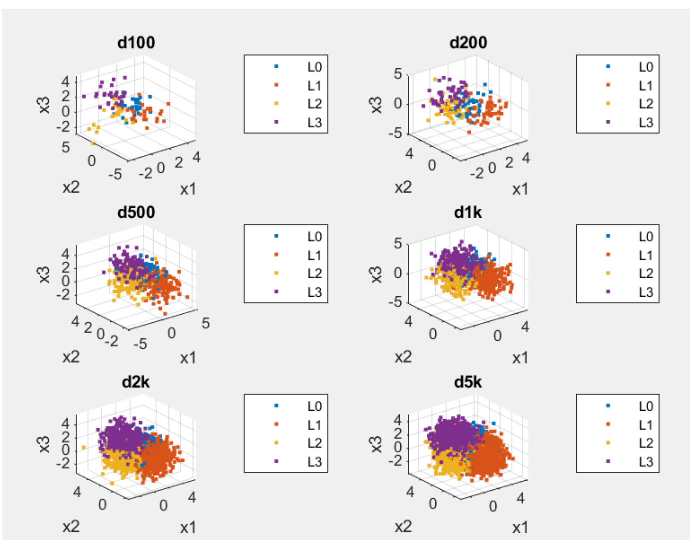
$\boldsymbol{m}1$=[2.6 0 0],$\boldsymbol{C}1$=[1.04 0.05 0.01, 0.03 1.04 0.01, 0.01 0.01 1]

$\boldsymbol{m}2$=[0 2.6 0],$\boldsymbol{C}2$=[1.06 0.08 0.05 , 0.06 1.05 0.07 , 0.06 0.05 1.05]

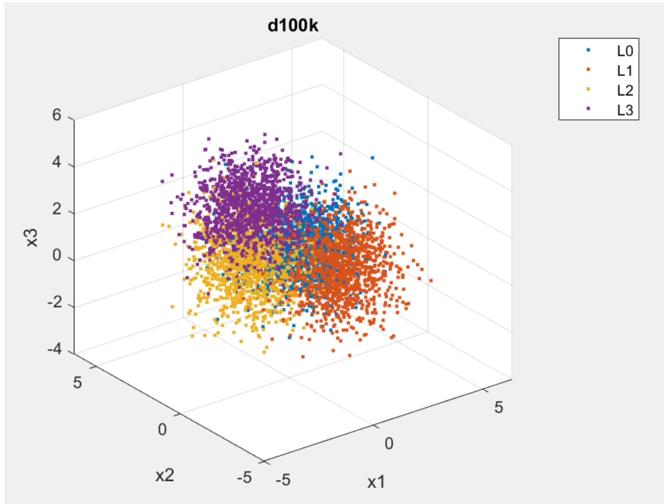
$\boldsymbol{m}3$=[0 0 2.6],$\boldsymbol{C}3$=[1.03 0.05 0.03 , 0.03 1.05 0.04 ,0.02 0.03 1.02]

A 2-layer MLP with one hidden and one output later was specified and implemented. The output layer was a "softmax" function as is the default for the Matlab "patternnet" function that was implemented. Softmax function is a more generalized logistic activation function which is used for the multiclass classification. Softmax ensure all outputs are positive and add up to 1. The best number of perceptrons for your custom problem will be selected using cross-validation. Additionally, in the problem a smooth-ramp style activation function was activated . A sigmoid function can cause a neural network to get stuck at the training time. RelLu is half rectified from bottom and is a monotonic function.

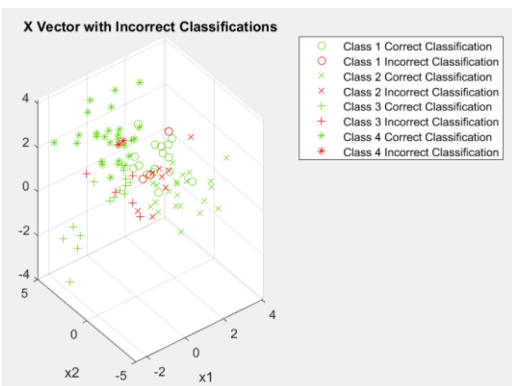
Below are the figures for training datasets with 100, 200, 500, 1000, 2000, and 5000 samples were generated and validation of test dataset with 100,000 samples was generated.



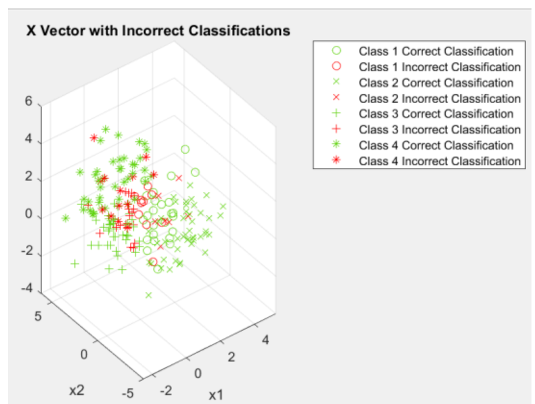
Training data set



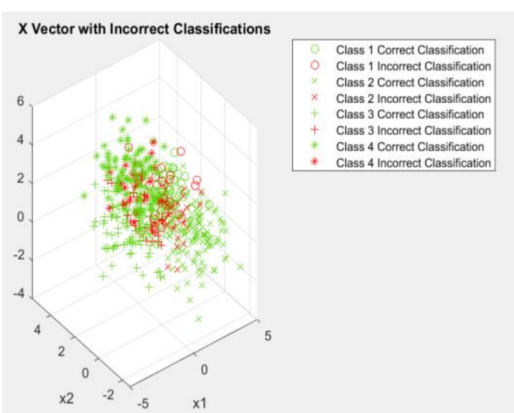
Validation Dataset 100k samples



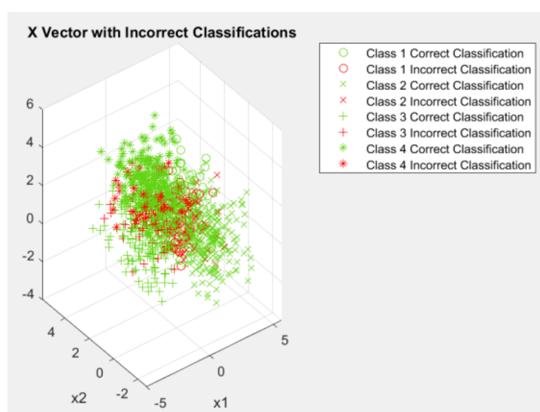
Classification of the 100 samples



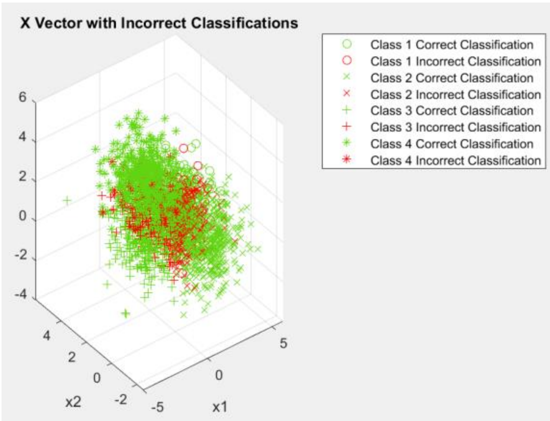
Classification of 200 samples



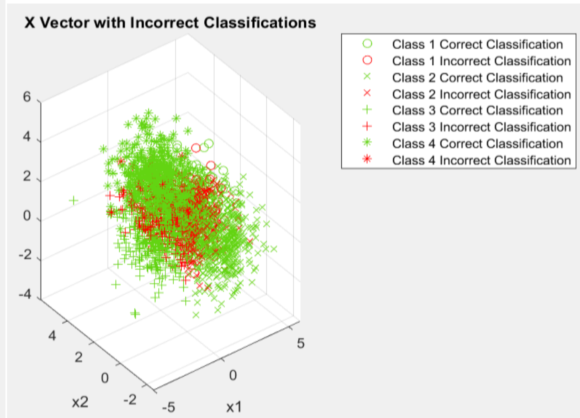
Classification of the 500 samples



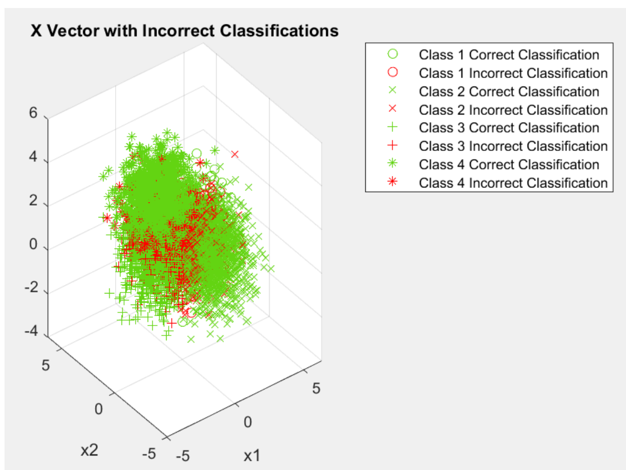
Classification of 1000 samples



Classification of the 2000 samples



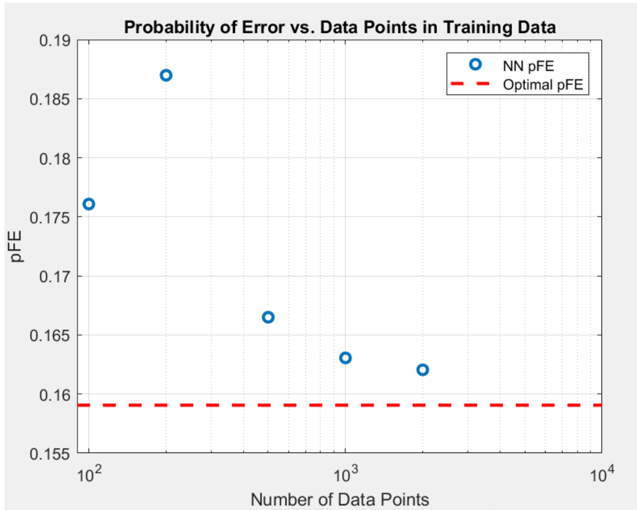
Classification of 5000 samples



Classification of the 100k samples

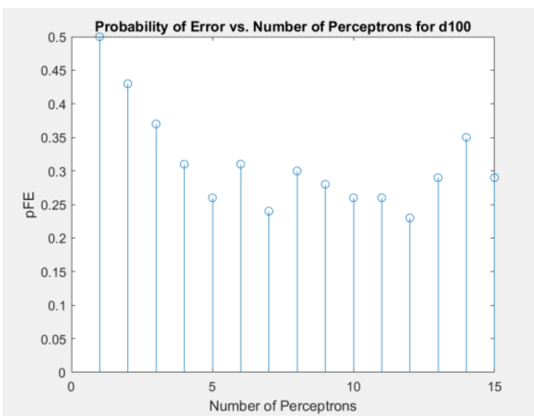
For each training dataset 10-fold cross validation was performed to determine the optimal number of perceptron for the MLP model. In 10k fold cross validation would perform the fitting procedure a total of ten times, with each fit being performed on a training set consisting of 90% of the total training set selected at random, with the remaining 10% used as a hold out set for validation . The optimal number was the one that resulted in the minimum probability of error across the cross validation runs. Once the number of perceptron was selected a final model was then trained on the entire training dataset. Finally, this trained model was evaluated using the test dataset and the probability of error was calculated as the metric of model performance.

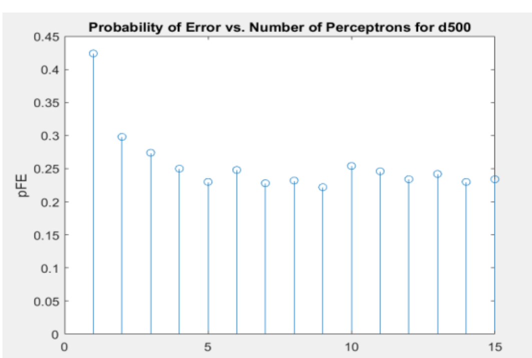
Below figure shows the results of this procedure. As can be seen in the plot, the overall probability of error is well correlated with the size of the training dataset. As the size of the dataset increases the probability of error decreases and approaches the optimal probability of error as estimated using the true pdf of the underlying data. This demonstrates that as the quantity of training data increases the model estimate is able to be improved resulting in more accurate classifications.

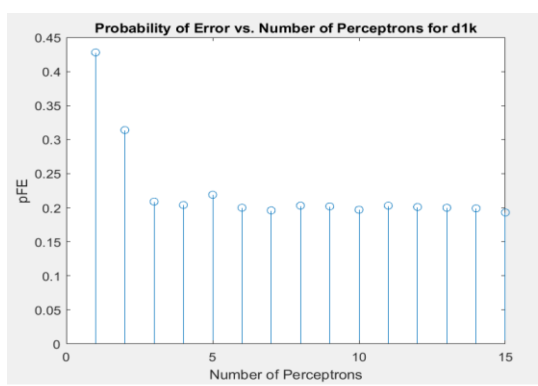


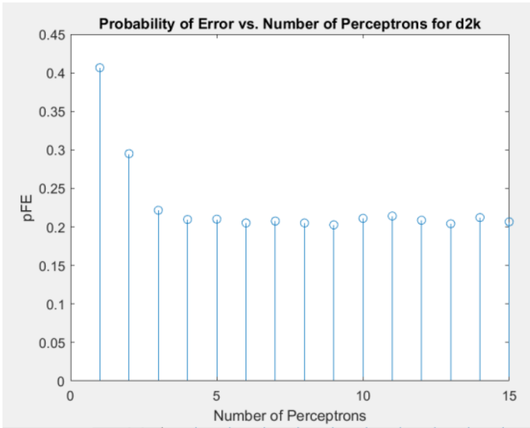
Probability of Error vs. Number of Data Points

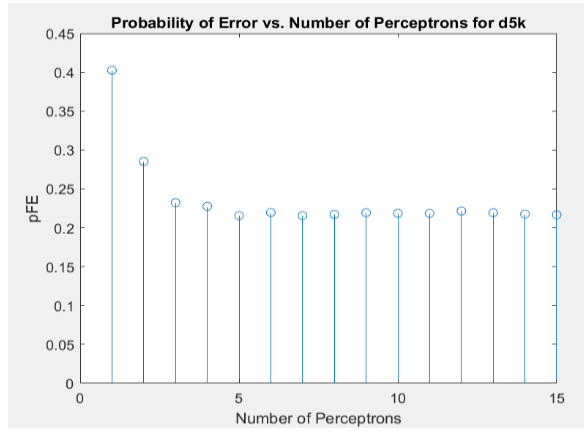
Probability error for various test datasets:











Below shows a plot of the optimal number of perceptron versus the number of data points in a dataset. Except for the data point for the 1000-point training set the optimal number of perceptron appears to increase as the size of the training dataset increases. This was expected since as the size of the dataset increases the complexity of the model can also increase. More data means more features than can be modelled and therefore model complexity increases. When the model is trained much its might reflect the incorrect probabilities too . Therefore the training of the model is done in such a way that it does not sows the noise and inappropriate probabilities .

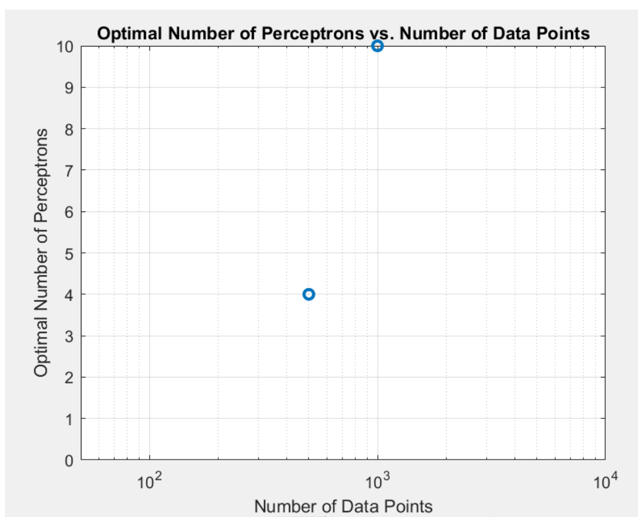
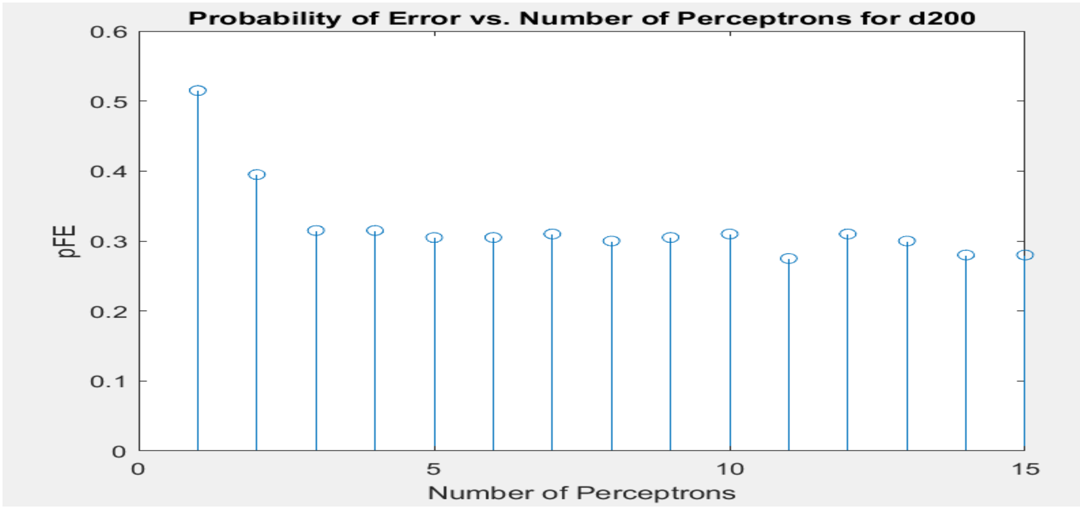


Fig. Optimal Number of Perceptron for a Given Dataset Size

Below shows a plot of the cross-validation results for the 200-sample training dataset. In the plot the probability of error is shown as a function of the number of perceptron. The probability of error starts very large for a single perceptron and then rapidly decreases as the number of perceptron increases. A minimum is identified at 11 and the probability error slowly tends to alter rom fmin to max as the number of perceptron is increased. While the optimal number of perceptron varied between training datasets the overall relationship between the probability of error and the number of perceptron generally followed this pattern.



Probability of Error vs. Number of Perceptron for 200 sample training dataset

Probability errors are obtained as below : They lie in the range of 10-20%. The gaussian mixtures is selected in such a way that it lies between the 10-20%

Optimal pFE, N=100: Error=11.00%

Optimal pFE, N=200: Error=14.00%

Optimal pFE, N=500: Error=17.40%

Optimal pFE, N=1000: Error=14.40%

Optimal pFE, N=2000: Error=15.25%

Optimal pFE, N=5000: Error=14.72%

Optimal pFE, N=100000: Error=15.91%

NN pFE, N=100: Error=17.61%

NN pFE, N=200: Error=18.70%

NN pFE, N=500: Error=16.65%

NN pFE, N=1000: Error=16.31%

NN pFE, N=2000: Error=16.20%

NN pFE, N=5000: Error=16.03%

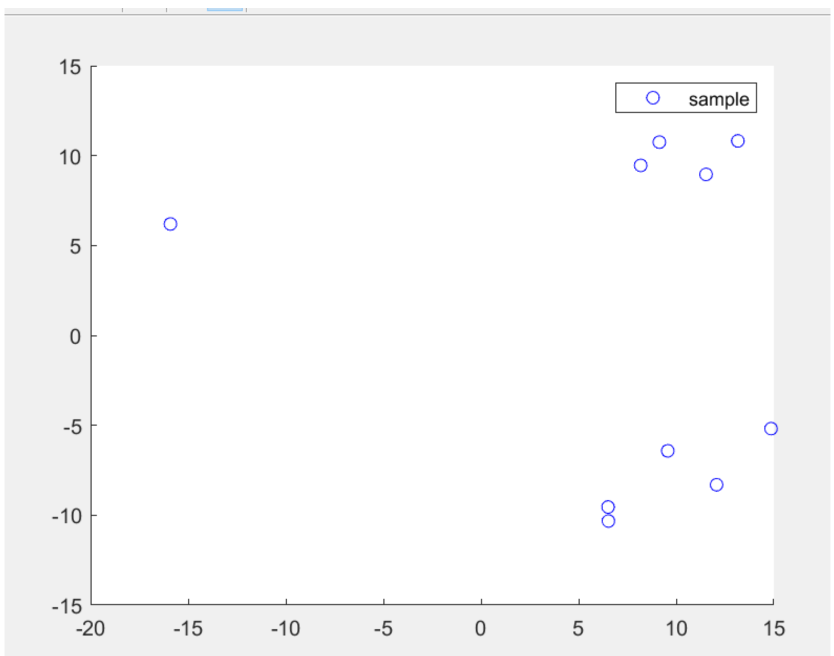
As observed the error probability decreases with the increase in the number of samples .

NN is the practically obtained probability errors .

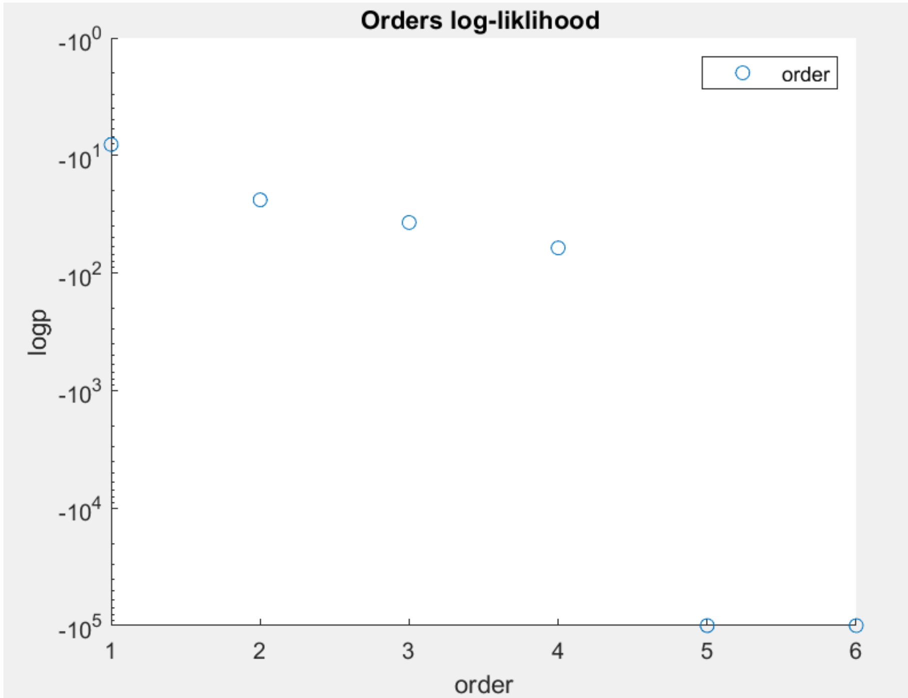
Question no 2:

Given question consist of the Gaussian Mixture Model as the true probability density function for 2-dimensional real-valued data synthesis. This GMM is having 4 components with different mean vectors, different covariance matrices, and different probability for each Gaussian to be selected as the generator for each sample.
Mean is given as : [-10 10 10 -10;10 10 -10 -10]
Covariances of 2-D matrix is given as :
Covariance 1 = [20 1;10 3];
Covariance 2 = [7 1;1 2];
Covariance 3 = [4 10;1 16];
Covariance 4) = [2 1;1 7];

The gaussian model is selected of a larger value in such a way that they are widely separated from each other so as to reflect that they are separated from each other. Moreover the K algorithm means once its run the groups are defined , any new data can be easily assigned to the correct group. Given in end we get the cluster that is visible in each of the GMM distribution of each samples.
For N=10 the GMM data generated as below

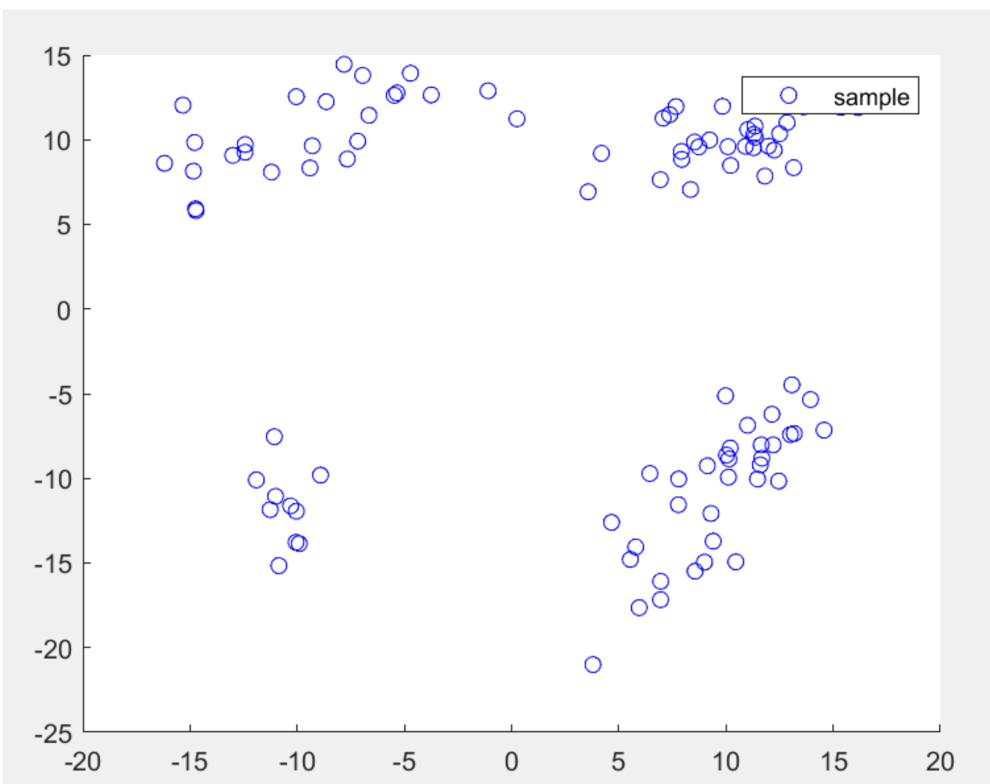


Gaussian distribution of the samples N=10

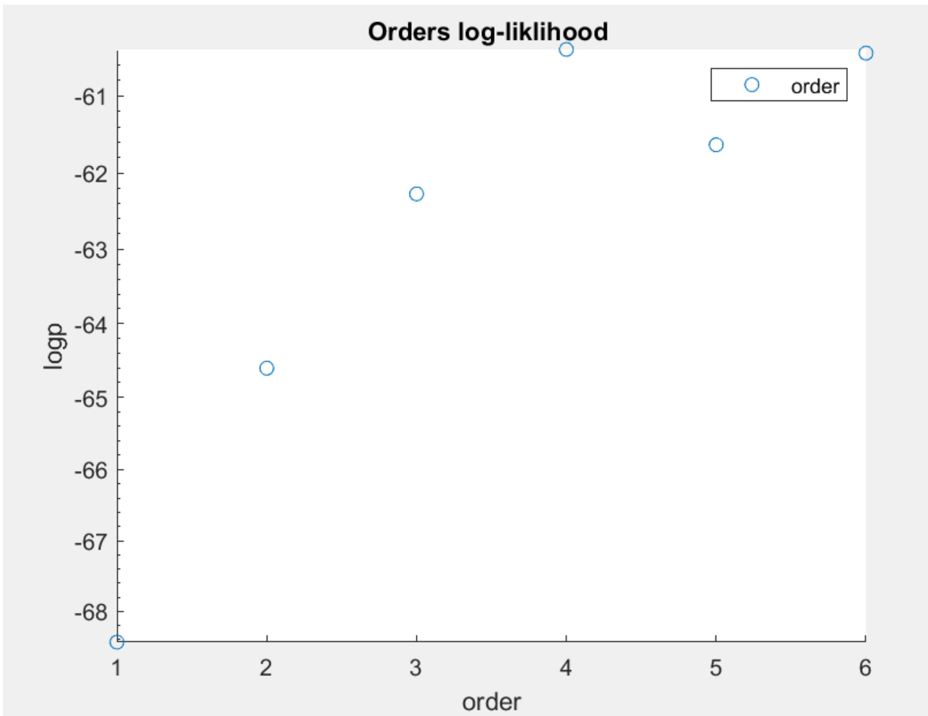


Negative Log-likelihood for the GMM Model N=10

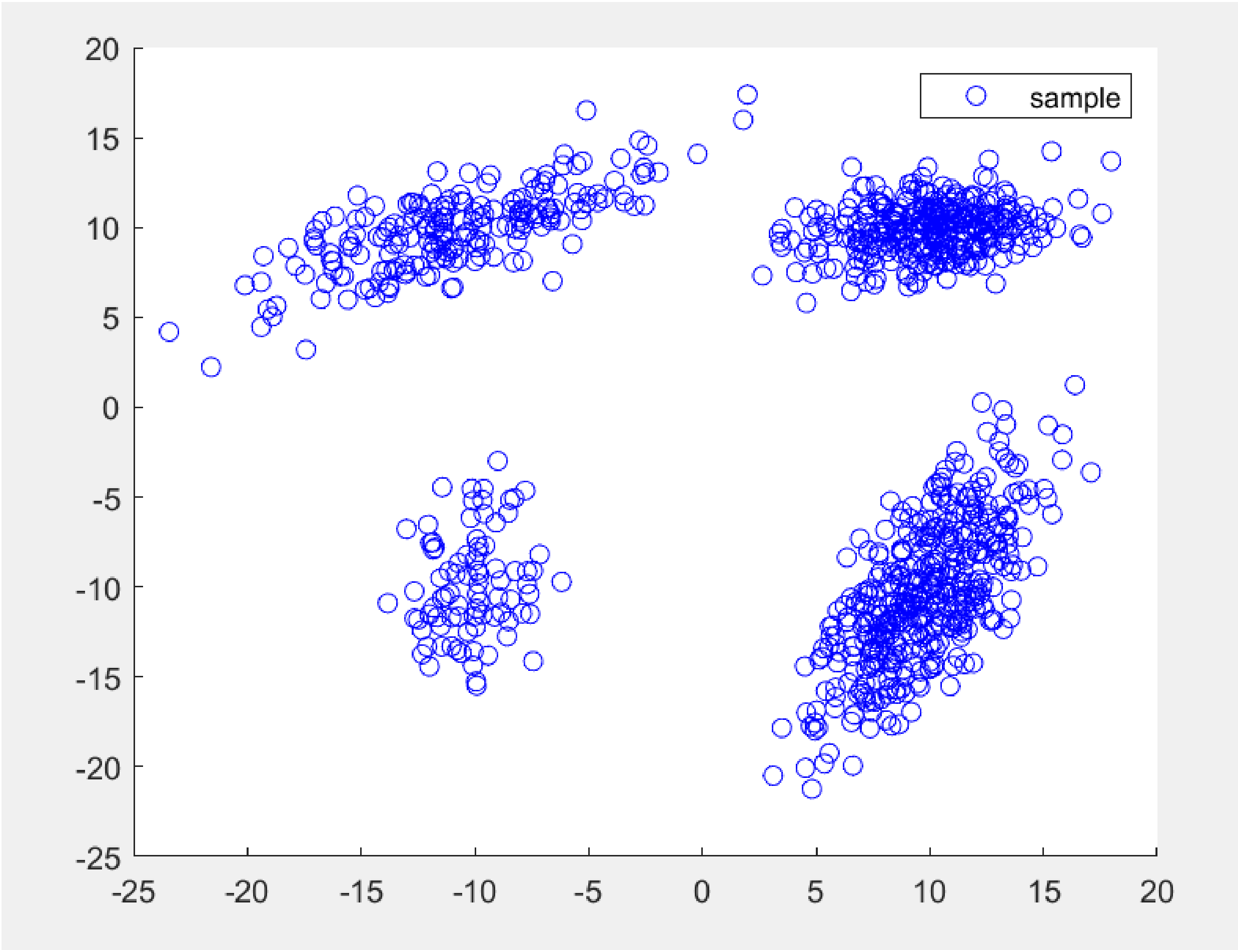
N= 100



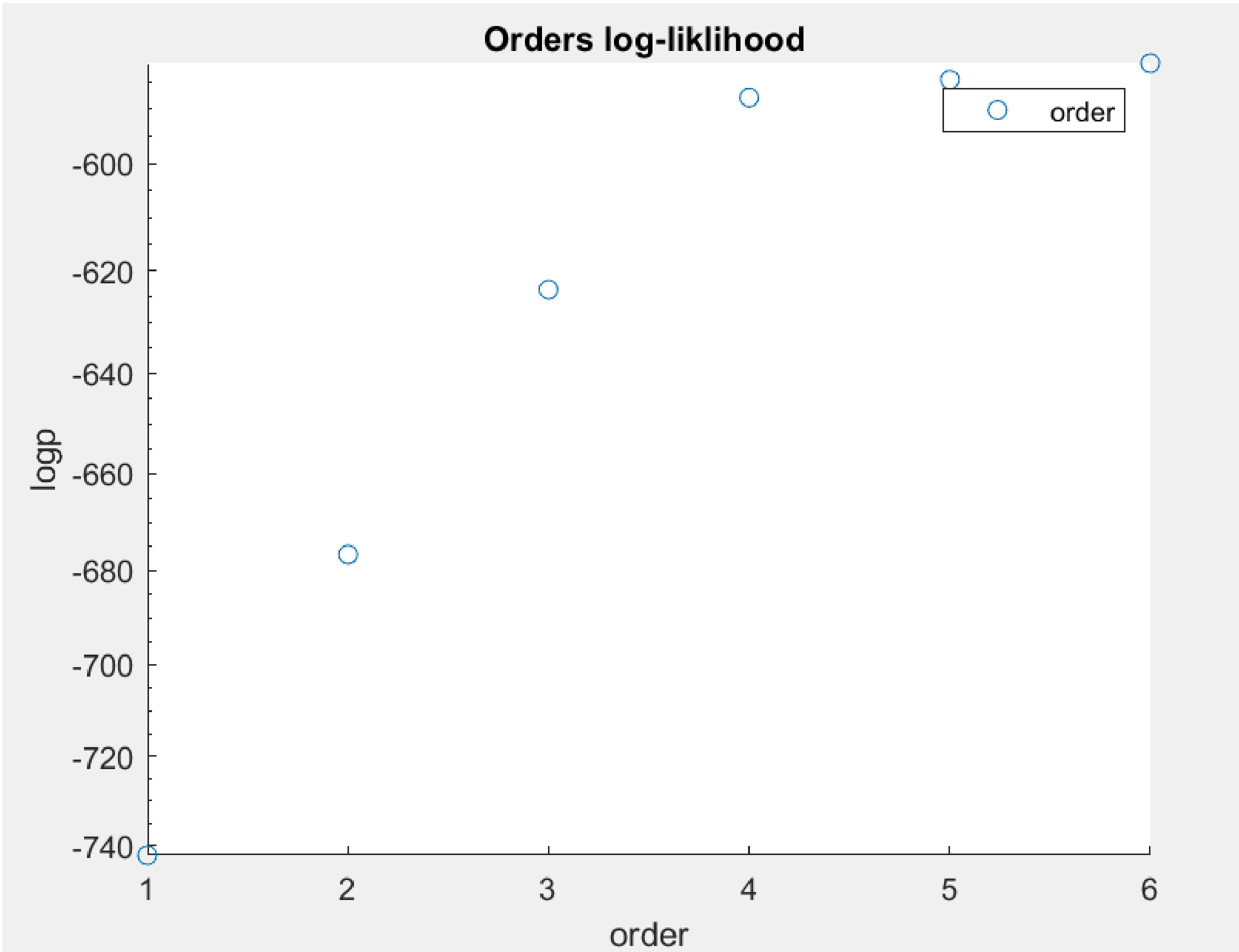
Gaussian distribution of the samples N=100



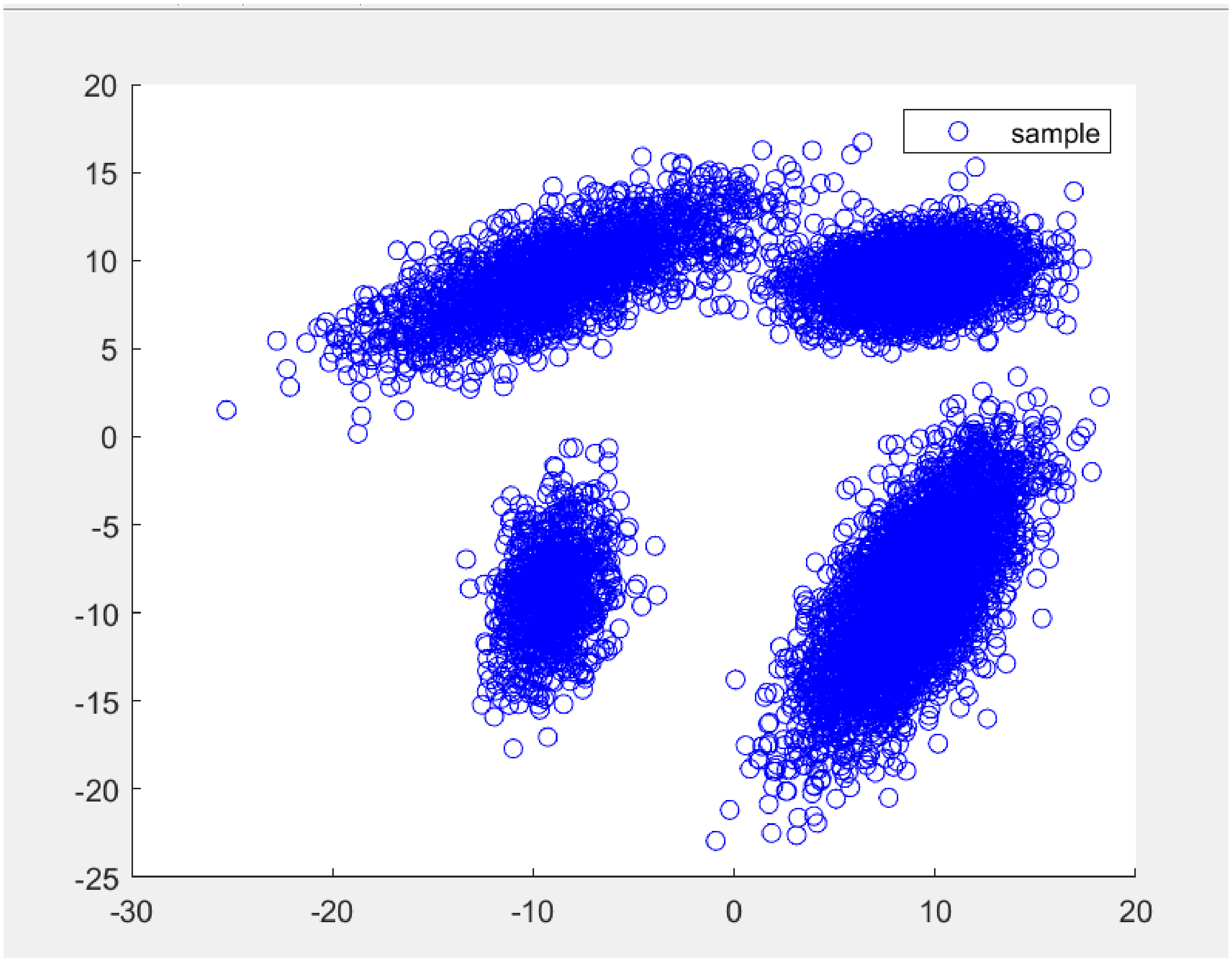
Negative Log-likelihood for the GMM Model N=100



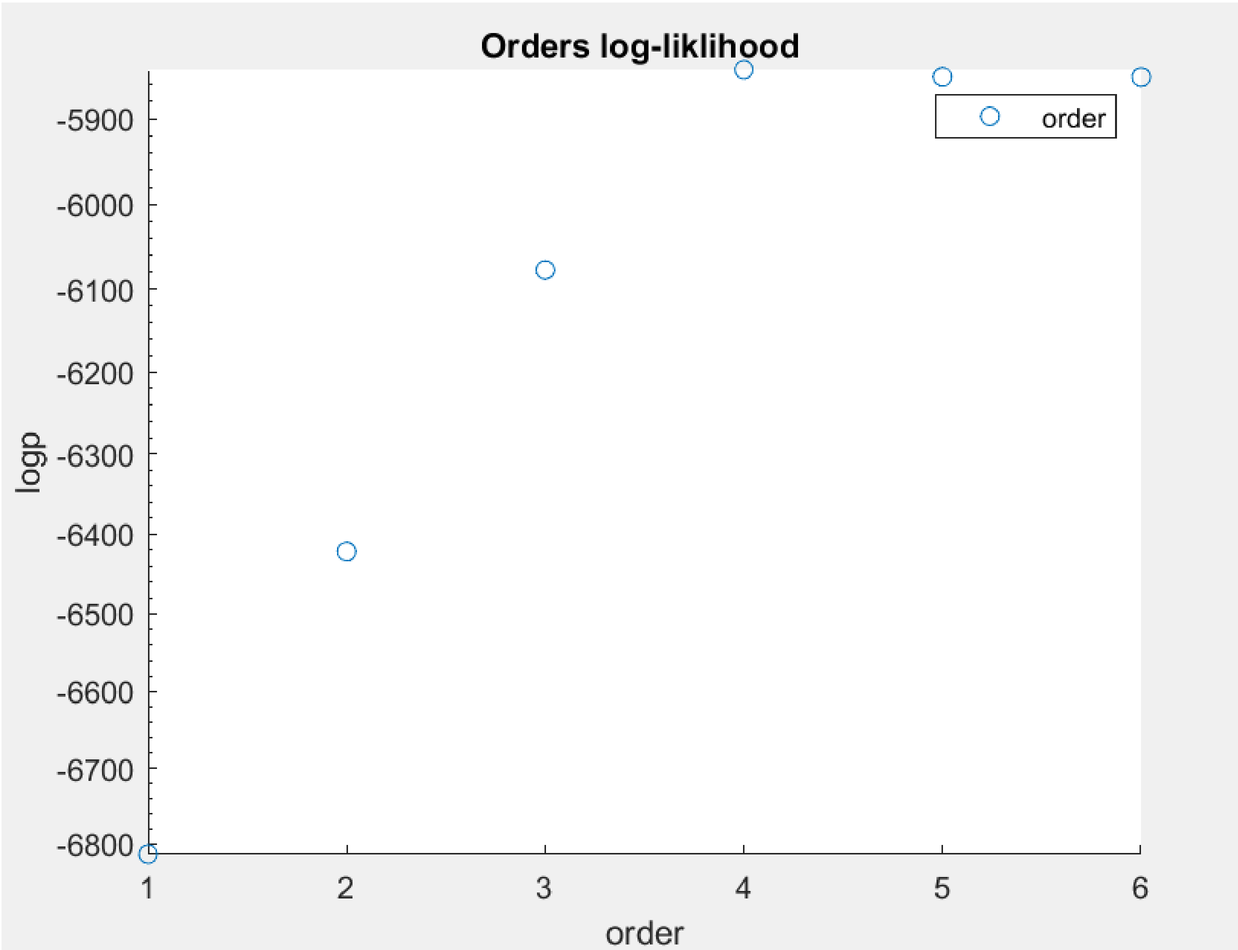
Gaussian GMM distribution of the samples N=1000



Negative Log-likelihood for the GMM Model N=1000



Gaussian GMM distribution of the samples N=10000



Negative Log-likelihood for the GMM Model N=10000

Conclusion :

The log likelihood indicates number of component =4 is a good model for this data . After the component 6 the it stops . Moreover at component=4 the graph quickly plateaus around the 4 . Hence we can consider it the best component for this gaussian mixture model .
As we see for the 10 to 100 samples the graph does not plateaus but from 1000 it starts to come becomes contact till the 6 point and when samples are increased it start to flatten and becomes constant from the 4^{th} component .

Appendix code 1: MATLAB

```
clc
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=2.6;
```

```
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*[1 0 0]';
RandSig=SigmaScale*rand(dimensions,dimensions);
Sigma.L1(:,:,1)=RandSig*RandSig'+eye(dimensions);
mu.L2=muScale*[0 1 0]';
RandSig=SigmaScale*rand(dimensions,dimensions);
Sigma.L2(:,:,1)=RandSig*RandSig'+eye(dimensions);
mu.L3=muScale*[0 1 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 [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 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
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
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 [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
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This function performs the cross validation and model selection
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
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
```

Appendix Question 2 code : Matlab

```
clear all,
close all,
N = 10000;
%N=1000; % taking the values and trying each time with the N as
%10,100,1000and 1000
%N=100;
%N=10;
delta = 1e0; % tolerance for EM stopping criterion
regWeight = 1e-10; % regularization parameter for covariance estimates
%Replicating it 30 times
% Generate samples from a 4-component GMM
alpha_true = [0.2,0.3,0.4,0.1];
mu_true = [-10 10 10 -10;10 10 -10 -10];
Sigma_true(:,:,1) = [20 1;10 3];
Sigma_true(:,:,2) = [7 1;1 2];
Sigma_true(:,:,3) = [4 10;1 16];
Sigma_true(:,:,4) = [2 1;1 7];
x = randGMM(N,alpha_true,mu_true,Sigma_true);
figure(1);
figure(1),scatter(x(1,:),x(2,:),'ob'), hold on,
figure(1),legend('sample')
d = 2;
K = 10;
dummy = ceil(linspace(0,N,K+1));
for k = 1:K
    indPartitionLimits(k,:) = [dummy(k)+1,dummy(k+1)];
end

avgp = zeros(1,6);
for M = 1:6
    psum = zeros(1,10);
```

```
    for k = 1:K
        indValidate = [indPartitionLimits(k,1):indPartitionLimits(k,2)];
        xValidate = x(:,indValidate); % Using folk k as validation set
        if k == 1
            indTrain = [indPartitionLimits(k,2)+1:N];
        elseif k == K
            indTrain = [1:indPartitionLimits(k,1)-1];
        else
            indTrain = [[1:indPartitionLimits(k-1,2)],[indPartitionLimits(k+1,1):N]];
        end
        xTrain = x(:,indTrain); % using all other folds as training set
        Ntrain = length(indTrain);
        Nvalidate = length(indValidate);
        [alpha,mu,Sigma] = EMforGMM(Ntrain,xTrain,M,d,delta,regWeight);% determine
dimensionality of samples and number of GMM components
        p = zeros(1,Nvalidate);
        for j = 1:Nvalidate
            for i = 1:M
                p(j) = p(j) +
alpha(i)*evalGaussian(xValidate(:,j),mu(:,i),Sigma(:,:,i));
            end
            p(j) = log(p(j));
        end
        psum(k) = sum(p);
        dummy(k,M)=sum(p);

    end
    avgp(M) = sum(psum)/10;
    if (avgp(M)== -inf)
        avgp(M) = -1e5;
    end
end
%Below code is by trying the fitgmdsit  and Replicating by 30 times
%for m=1:6
%for i=1:K
    %[train, test]= kfld(X,i);%gives training and test data for a fold
   % model_gmm = fitgmdist(train,m);%fitting a gmm model on train data for m
components
   % prior=model_gmm.ComponentProportion;%priors
   % mean1=model_gmm.mu;%mean
   % mean=mean1';
   % cov=model_gmm.Sigma;%covariance
   % logLikelihood(i) = sum(log(evalGMM(test,prior,mean,cov)));
%end
%final_mean(m)=sum(logLikelihood)/10;%Final probability for all 6 Components
%end
figure(2),scatter([1,2,3,4,5,6],avgp),set(gca,'yscale','log'),
figure(2),legend('order'),title('Orders log-liklihood '),
xlabel('order'), ylabel('logp')

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

```
Converged = 0; % Not converged at the beginning
while ~Converged
    for l = 1:M
        temp(l,:) = repmat(alpha(l),1,N).*evalGaussian(x,mu(:,l),Sigma(:,:,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-alpha'));
    Dmu = sum(sum(abs(muNew-mu)));
    DSigma = sum(sum(abs(abs(SigmaNew-Sigma))));
    Converged = ((Dalpha+Dmu+DSigma)<delta); % Check if converged
    alpha = alphaNew; mu = muNew; Sigma = SigmaNew;
    t = t+1;

end
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) = 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 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
```