EECE5644 HW3

CONTENT

[Question 1 1](#_Toc86938451)

[Question 2 9](#_Toc86938452)

[Appendix code for Question 1 18](#_Toc86938453)

[Appendix code for Question 2 25](#_Toc86938454)

# 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.

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 used for implementation. In the problem a smooth-ramp style activation function was specified and it could be implemented by change the activation function like net.layers{1}.transferFcn='logsig'.

However there is not a built-in function of that type for “patternnet” in Matlab. Due to the challenges inherent in implementing it in the Matlab environment is too difficult(need to define it by myself), the default “tansig” function was used after asking the professor's permission.

For training datasets with 100, 200, 500, 1000, 2000, and 5000 samples were generated and for validation a test dataset with 100,000 samples was generated. Plots of the generated data are shown in Figure 1 and Figure 2.

图示

描述已自动生成

*Figure 1: Training Datasets*

图表, 散点图

描述已自动生成

*Figure 2: Validation Dataset*

For each training dataset 10-fold cross validation was performed to determine the optimal amount of perceptron for the MLP model. 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.

Figure 3 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.

图表, 散点图

描述已自动生成

*Figure 3: Probability of Error vs. Number of Data Points*

Figure 4 shows a plot of the optimal amount of perceptron versus the number of data points in a dataset. Except for the data point for the 1000-point training set the optimal amount 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 in a meaningful way. More data means more features than can be modelled and therefore model complexity increases.

图表, 散点图

描述已自动生成

*Figure 4: Optimal Amount of Perceptron for a Given Dataset Size*

Table 1 shows the optimal and the neutron network minimum probability of error with different datasets.

*Table 1: optimal and NN Probability of Error*

|  |  |  |
| --- | --- | --- |
|  | Optimal | NN |
| 100 | 13.00% | 16.62% |
| 200 | 12.00% | 14.01% |
| 500 | 12.00% | 14.13% |
| 1000 | 11.90% | 13.60% |
| 2000 | 12.55% | 13.41% |
| 5000 | 12.94% | 13.21% |
| 10000 | 13.11% | - |

Figure 5 below shows a plot of the cross-validation results for all training datasets. In the plot the probability of error is shown as a function of the amount 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 10 and the probability error slowly increases 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.

图表, 直方图

描述已自动生成图表, 直方图

描述已自动生成

图表, 直方图

描述已自动生成图表, 直方图

描述已自动生成

图表, 直方图

描述已自动生成图表, 直方图

描述已自动生成

*Figure 5: Probability of Error vs. Amount of Perceptron for training datasets*

Figure 6 and figure 7 show the X vectors with the correct and incorrect classification for training datasets and validate dataset.

图表, 散点图

描述已自动生成 图表, 散点图

描述已自动生成

图表, 散点图

描述已自动生成 图表, 散点图

描述已自动生成

图表, 散点图

描述已自动生成 图表, 散点图

描述已自动生成

*Figure 6: X vector with Classifications for training datasets*

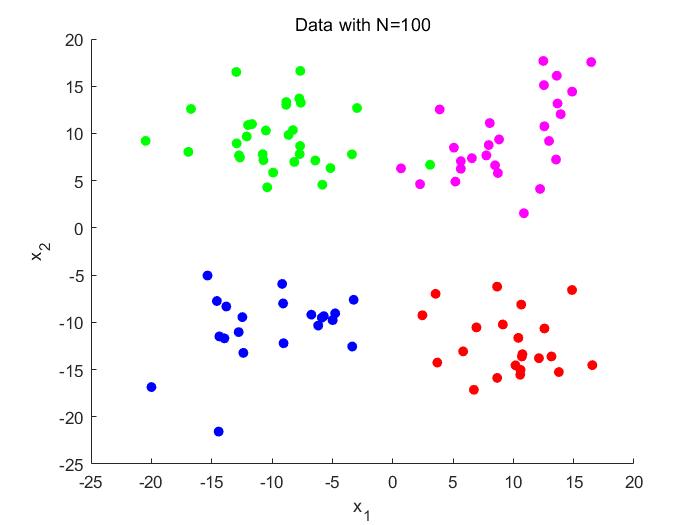
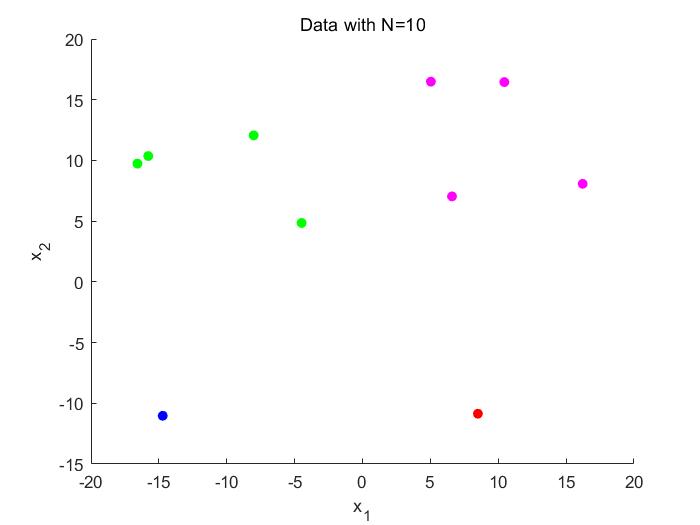
图表, 气泡图

描述已自动生成

*Figure 6: X vector with Classifications for validate dataset*

# Question 2

1. Class conditional PDFs and mixture coe\_cients:
2. Data with 10, 100, 1000 and 10k samples are generated using the GMM model described above. Figure 7 below shows the realization of the models with different samples.



图表, 散点图

描述已自动生成图表, 散点图, 气泡图

描述已自动生成

*Figure 7: Data visualization*

1. With number of samples=1000 or 10k, log likelihood and BICs indicate number of components=4 is a good model for this data. What’s more, when n components=4, the score quickly falls and plateaus when Components=1,2,3,5,6. Considering Occam’s razor, number of components=4 is obviously the best option.

图表, 折线图

描述已自动生成图表, 折线图

描述已自动生成 图表, 折线图

描述已自动生成 图表, 折线图

描述已自动生成

*Figure 8: Training log\_likelihoods for all datasets*

图表, 折线图

描述已自动生成 图表, 折线图

描述已自动生成

图表, 折线图

描述已自动生成 图表, 折线图

描述已自动生成

*Figure 9: Validation log\_likelihoods for all datasets*

图表, 折线图

描述已自动生成 图表, 折线图

描述已自动生成

图表, 折线图

描述已自动生成 图表, 折线图

描述已自动生成

*Figure 10: Training BICs for all datasets*

图表, 折线图

描述已自动生成图表, 折线图

描述已自动生成

图表, 折线图

描述已自动生成 图表, 折线图

描述已自动生成

*Figure 11: Validation BICs for all datasets*

1. I am not able to get the algorithm converging in all cases, despite adding Gaussian noise to the training set, re-initializing mu values multiple times throughout, or adjusting the original distribution itself. Therefore, the algorithm has be limited for each combination of parameters. This does not provide a solution for the convergence issue, but there still some data to show the performance of the algorithm.

Though we expect that the model order of 4 gets selected higher number of times as data increases, it does not seem to be happening in this case. possible reason could be that since we are using bootstrapping, the training sets thus generated might not have covered data such that it can capture the variation. Or secondly, the results could also depend on the size of training and validations sets generated. In order to check if the correct model order is selected for more data or if it also depends on the size of training and validation set, the following table is plotted.

Figure 12 below shows the results of running the algorithm for the first experiment and keeping track of when each component number was selected as the winner for the tenth and twentieth experiment(figure13 and figure 14) and the result after all 30 experiments(figure15).

图表

描述已自动生成

*Figure 12: GMM Component Selection for 1st experiment*

As shown in the figure12, 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 (2 component GMMs). 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. With 100 and 1000 sample data sets, however, the distribution was normal-shaped, with the center (winner in most iterations) being a 4-component GMM. This shows that with more points at disposal to form the training set, the algorithm can pick the correct component number more often.

图表, 条形图

描述已自动生成

*Figure 13: GMM Component Selectionfor 10th experiment*

*Table 2: Rate of order selection for 10th experiment*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 |
| 10 | 10.00% | 0.00% | 10.00% | 10.00% | 30.00% | 40.00% |
| 100 | 30.00% | 20.00% | 10.00% | 10.00% | 0.00% | 30.00% |
| 1000 | 10.00% | 10.00% | 20.00% | 40.00% | 0.00% | 20.00% |
| 10k | 20.00% | 10.00% | 30.00% | 0.00% | 20.00% | 20.00% |

图表, 条形图

描述已自动生成

*Figure 14: GMM Component Selectionfor 20th experiment*

*Table 2: Rate of order selection for 20th experiment*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 |
| 10 | 10.00% | 10.00% | 5.00% | 15.00% | 35.00% | 25.00% |
| 100 | 30.00% | 15.00% | 5.00% | 20.00% | 15.00% | 15.00% |
| 1000 | 15.00% | 5.00% | 20.00% | 40.00% | 5.00% | 15.00% |
| 10k | 35.00% | 5.00% | 15.00% | 15.00% | 10.00% | 20.00% |

From the above figures and tables it can be noticed that though there is not necessarily a pattern with increase in size of training/validations sets, there is a clear pattern with an increase in size of actual data from which we generate training/validation sets. This is as we expected - as the data increases, the estimation gets better and the frequency of correct model order selection increases.

图表, 条形图

描述已自动生成

*Figure 15: GMM Component Selection after 30 experiments*

*Table 2: Rate of order selection*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 |
| 10 | 10.00% | 16.67% | 6.67% | 16.67% | 26.67% | 23.33% |
| 100 | 23.33% | 16.67% | 6.67% | 20.00% | 16.67% | 16.67% |
| 1000 | 16.67% | 6.67% | 16.67% | 36.67% | 6.67% | 16.67% |
| 10k | 33.33% | 10.00% | 13.33% | 10.00% | 13.33% | 20.00% |

For the built-in MATLAB function, for each iteration of the EM algorithm, two things had to be provided the training set, and the number of components. The function then fits a Gaussian mixture with the selected number of components to the training data using the EM algorithm and reports back on the α，μ and Σ values for the distributions. When using the built-in function, to ensure the parameters are not constrained in any way, covariance did not be selected to be diagonal or shared between the distributions (covariance remained independent and full for each component within a GMM). I also chose to add a small regularization term (­1-10) to ensure that all covariance matrices are positive-definite as the algorithm is being executed.

Figure 15 upon summarizes the results of running the EM algorithm for 30 experiments. Table 2 shows the rate of order selection. As can be observed, when 10 data points were used to form the full dataset, the algorithm seemed to favor the higher components GMM. When 100 data points were used to form the full dataset, the algorithm seemed to even out. when 1000 and 10k data points were used to form the full dataset, the algorithm seemed to favor the higher components GMM too, especially for 4-components. This may due to the original 4 components of the true GMM has more close prior value and the center of the mixture model is less overlap. Since MATLAB does not make their function files public, it’s different to look through what makes MATLAB'S implementation and the EM algorithm and the one we discussed in class different. However, the non-built-in function from the previous part seems to have a better performance with the same problem better since it was able to pick the true component number more times.

\

Appendix code for Question 1

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
%EECE5644 Fall 2021  
% Wang Yinan 001530926 | HW3  
%%=========================Question 1=========================%%  
% Code help and example from Prof.Deniz  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
clear all;close all;clc;  
  
%%=========================Setup=========================%%  
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 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);  
    title([dTypes{ind}]);  
    legend 'show';  
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,dTypes{ind});  
    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,dTypesind)  
% 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 for ' dTypesind]);  
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) ...  
            'for ' dTypesind]);  
        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 code for Question 2

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
%EECE5644 Fall 2021  
% Wang Yinan 001530926 | HW3  
%%=========================Question 2=========================%%  
% Code help and example from Prof.Deniz  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
clear all;close all;clc;  
  
% variance  
n=2;  
alpha\_true=[0.20,0.30,0.23,0.27];  
% mu\_true=[10 -10 -10 10;-10 10 -10 10];  
mu\_true(:,1) = [10;-10];  
mu\_true(:,2) = [-10;10];  
mu\_true(:,3) = [-10;-10];  
mu\_true(:,4) = [10;10];  
Sigma\_true(:,:,1) = [15 1;1 15];  
Sigma\_true(:,:,2) = [17 3;3 17];  
Sigma\_true(:,:,3) = [19 5;5 19];  
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);  
  
% 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  
  
  
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','filled');  
    hold on  
    scatter(x(1,label==2),x(2,label==2),'g','filled');  
    hold on  
    scatter(x(1,label==3),x(2,label==3),'b','filled');  
    hold on  
    scatter(x(1,label==4),x(2,label==4),'m','filled');  
    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,'.b');   
    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,'.b');  
    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,'rx');  
    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  
  
  
%%=========================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)  
%PerformsEMalgorithmtoestimateparametersandevalueteperformance  
%oneachdatasetBtimes,with1throughMGMMmodelsconsidered  
B=10;M=6;%repetitionsperdataset;maxGMMconsidered  
perf\_array=zeros(B,M);%savespaceforperformanceevaluation  
%Testeachdataset10times  
for b=1:B  
    %Pickrandomdatapointstofilltrainingandvalidationsetand  
    %addnoise  
    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  
        %Non􀀀Built􀀀In:runEMalgorithtoestimateparameters  
        %[alpha,mu,sigma]=EMforGMM(m,trainset,setsize,valset);  
        %Built􀀀Infunction:runEMalgorithmtoestimateparameters  
        GMModel=fitgmdist(train\_set',M,'RegularizationValue',1e-10);  
        alpha=GMModel.ComponentProportion;  
        mu=(GMModel.mu)';  
        sigma=GMModel.Sigma;  
        %Calculatelog􀀀likelihoodperformancewithnewparameters  
        perf\_array(b,m)=sum(log(evalGMM(val\_set,alpha,mu,sigma)));  
    end  
end  
% Ca l cul a t e average per formance f o r each M and f i n d be s t f i t  
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)];  
%{  
% when the samples' num is small(like 10)  
% there could be non-generated class  
if ismember(0,Nc)  
    % find non-generated class  
    a=find(Nc==0);  
    % add 1  
    Nc(a)=1;  
    % which class's num is the max  
    b=find(Nc==max(Nc));  
    % minus 1 to keep the total nums  
    Nc(b)=Nc(b)-1;  
    % find the max-class position in label  
    c=find(label==b);  
    % change the first position to non-generated class  
    label(c(1))=a;  
end  
%}  
% 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  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%