EECE-5644

INTRO TO MACHINE LEARNING AND PATTERN RECOGNITION

ASSIGNMENT 3

SAI PRASASTH KOUNDINYA GANDRAKOTA
NUID: 002772719

QUESTION 1

PROCESS

DATA GENERATION

Given that there are 4 classes, and the input X is 3-dimensional, we generate training sets of sizes 100, 200, 500, 1000, 2000 and 5000 along with a test set of size 10000 using the below mean and covariance matrices (varied through trial and error based on keeping the error probability between 10-20%) and the Gaussian Mixture Model with uniform priors:

$$m_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} ; \quad m_2 = \begin{bmatrix} 0 \\ 0 \\ 5 \end{bmatrix} ; \quad m_3 = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix} ; \quad m_4 = \begin{bmatrix} 5 \\ 0 \\ 0 \end{bmatrix}$$

$$C_1 = \begin{bmatrix} 1 & 0 & -2 \\ 0 & 1 & 0 \\ -2 & 0 & 12 \end{bmatrix} ; \quad C_2 = \begin{bmatrix} 6 & 0 & 0 \\ 0 & 0.3 & 0 \\ 0 & 0 & 0.3 \end{bmatrix} ; \quad C_3 = \begin{bmatrix} 1 & 0 & -2 \\ 0 & 0.3 & 0 \\ -2 & 0 & 12 \end{bmatrix} ; \quad C_4 = \begin{bmatrix} 6 & 0 & 2 \\ 0 & 1 & 0 \\ 2 & 0 & 12 \end{bmatrix}$$

$$\text{Uniform Priors} \rightarrow \left[0.25, 0.25, 0.25, 0.25\right]$$

NEURAL NETWORK ARCHITECTURE

A 2-layer MLP, with one layer of P hidden perceptrons with the ELU activation function and an output layer with the softmax function applied, is created. Then using 10-fold cross validation, where we split the data into 10 folds, use each fold as a validation data and the rest of the folds apart from the validation data as training data, and then find the best number of perceptrons for the hidden layer (between 1 and 10) based on the minimum classification error probability.

TRAINING THE NEURAL NETWORK

Having identified the optimum number of perceptrons for each training set, we then train 6 different MLPs each with their corresponding data set according to the minimum cross-entropy loss estimation and identify the best weights and biases over 10 reinitializations.

TESTING THE NEURAL NETWORK

Using the respective weights and biases for each MLP we then predict the labels of the test data set and estimate the probability of error along with the confusion matrices, approximating the MAP classification rule.

THEORETICALLY OPTIMUM CLASSIFIER

We train an ideal Bayesian Classifier using the below rule:

Error = Loss Matrix * Class Posteriors

$$E(Y=c|X) = LM * P(L=c|X)$$

where $X \rightarrow$ input
$Y \rightarrow$ prediction
$L \rightarrow$ actual
$C \rightarrow$ classes $(0,1,2,3)$
$E \rightarrow$ error probability
$LM \rightarrow$ loss matrix

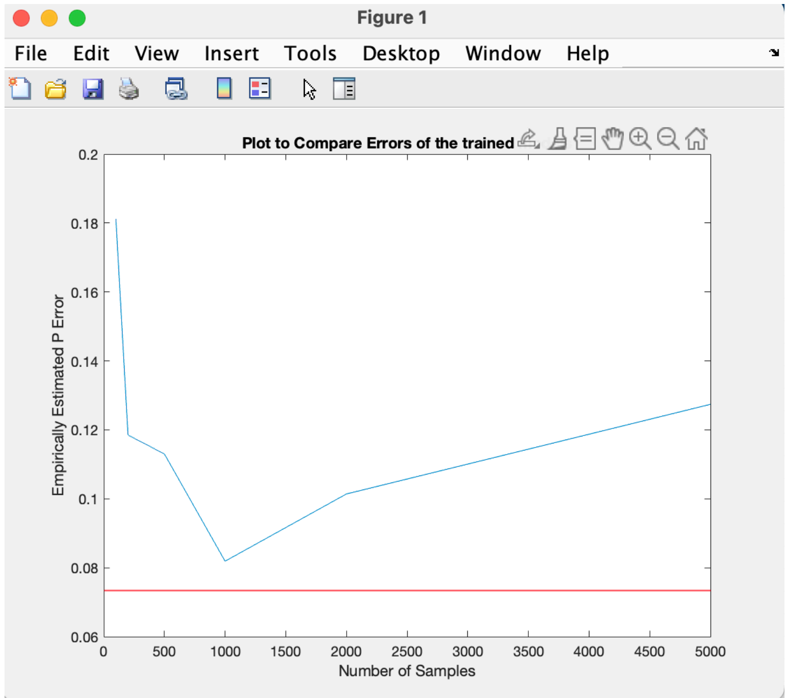
The prediction Y is obtained by minimizing the risk of choosing each class.

RESULTS

Each MLP model along with the optimum number of perceptrons in the hidden layer and their corresponding minimum classification error, final probability error on the test data set and confusion matrix are given below. We see that as the number of perceptrons increases, the error decreases and saturates at a point where we can select the optimum number.

```
MLP MODEL 1 (100 training samples)
----------------------------------
Optimum number of perceptrons = 8
Min Classification Error = 0.160000
Probability of Error = 0.181200
Confusion Matrix =
       4807         474           9         638
         77         776           0          37
        277           1         565          11
        163         124           1        2040

MLP MODEL 2 (200 training samples)
----------------------------------
Optimum number of perceptrons = 9
Min Classification Error = 0.120000
Probability of Error = 0.118500
Confusion Matrix =
       5581         206          66          75
        320         453           0         117
          3           0         825          26
        318          48           6        1956

MLP MODEL 3 (500 training samples)
----------------------------------
Optimum number of perceptrons = 7
Min Classification Error = 0.146000
Probability of Error = 0.113000
Confusion Matrix =
       5368         443          38          79
         98         729           0          63
          0           0         854           0
        293         113           3        1919

MLP MODEL 4 (1000 training samples)
----------------------------------
Optimum number of perceptrons = 10
Min Classification Error = 0.211000
Probability of Error = 0.081900
Confusion Matrix =
       5569         256           1         102
        102         756           0          32
          1           0         853           0
        257          68           0        2003

MLP MODEL 5 (2000 training samples)
----------------------------------
Optimum number of perceptrons = 8
Min Classification Error = 0.394000
Probability of Error = 0.101400
Confusion Matrix =
       5870          16           1          41
        498         272           0         120
          3           0         851           0
        314          20           1        1993

MLP MODEL 6 (5000 training samples)
----------------------------------
Optimum number of perceptrons = 10
Min Classification Error = 0.400600
Probability of Error = 0.127400
Confusion Matrix =
       5871          25           3          29
        566          23           0         301
          0           0         854           0
        321          28           1        1978
```

In the graph shown below we plot the probability error obtained from each MLP compared to the theoretically optimal classifier. We see that the models have higher error at lower training set sizes but higher training set sizes could also be overfit to throw off the model.

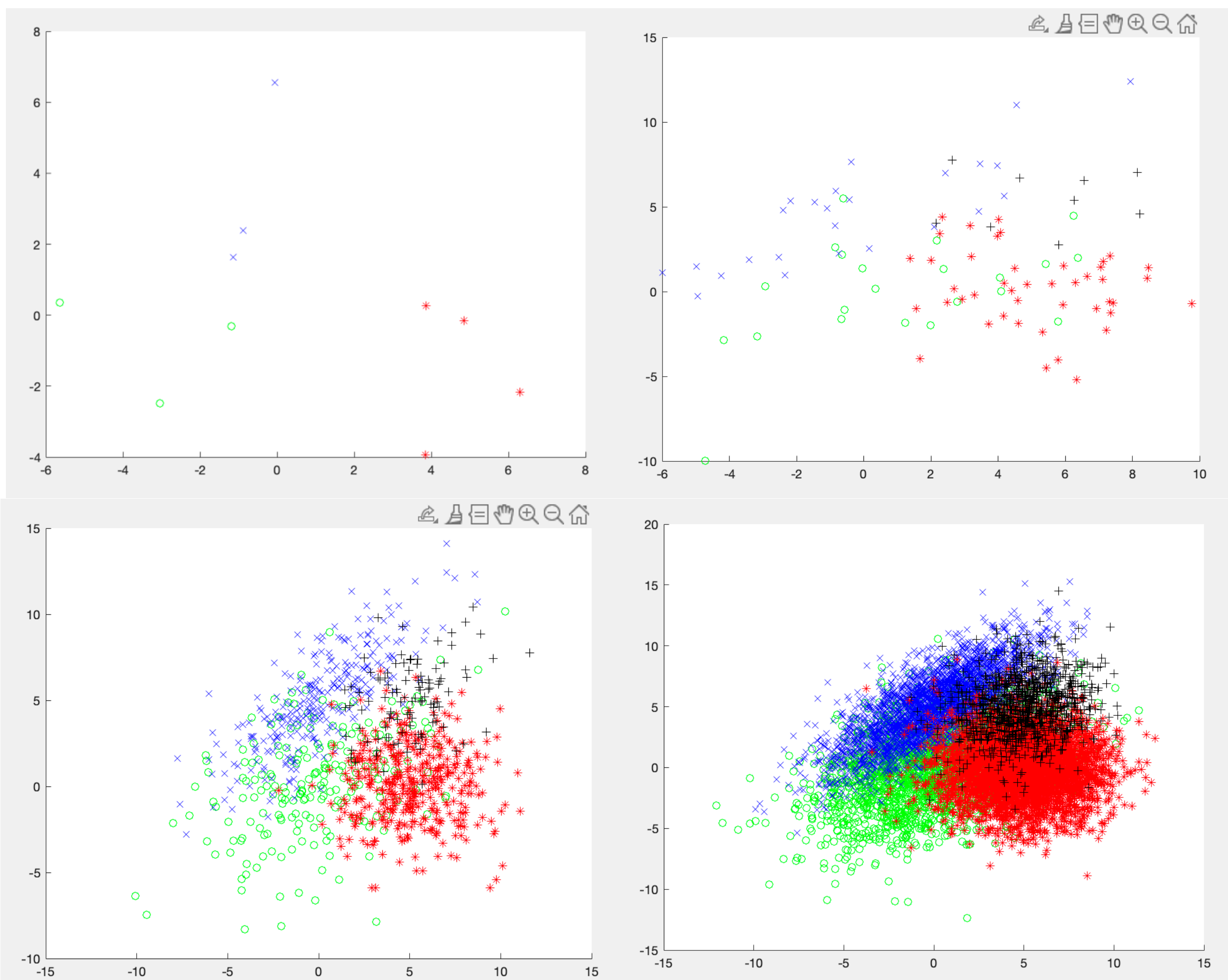


QUESTION 2

PROCESS

DATA GENERATION

Datasets comprising of 10, 100, 1000, and 10000 samples were created using a Gaussian Mixture Model that generates 2-dimensional data with four distinct classes, each having different class priors. The visual representations of the data distributions for these datasets are shown below:



The means and covariance matrices are given below:

M1 = [0,0]; M2 = [0,5]; M3 = [5,0]; M4 = [5,5];
S1 = [10,4;4,10]; S2 = [8,6;6,8]; S3 = [5,0;0,5]; S4 = [4,1;1,5];

GAUSSIAN MIXTURE MODEL

The Gaussian Mixture Model is initialized with a random mean, variance and prior probabilities and using Gaussian Components ranging from 1 to 6 we fit the GMM distribution onto the training data and then calculate the log likelihood of the validation data. The max iterations was set at 3000, the tolerance set to 1 x e^{-6} and the regularization value was given as 0.01.

TEN-FOLD CROSS VALIDATION

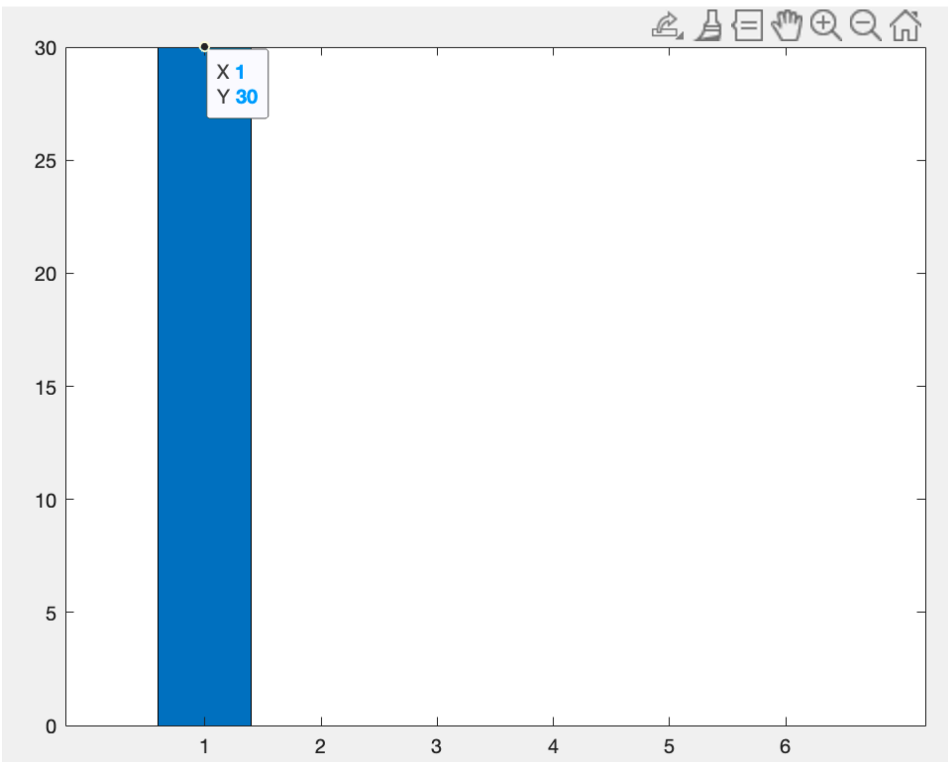
Each GMM order with Gaussian components ranging from 1 to 6 is cross validated by splitting the data into 10 folds, taking each fold as a validation fold and the remaining as the training folds, we fit the GMM distribution onto the training data and then calculate the log likelihood of the validation data. The likelihood across all 10 folds is averaged to give the score for that model.

MODEL ORDER SELECTION

The model with the best score computed above will be selected as the best GMM model. This is repeated 30 times for each data set and the selection rates for each model are tallied.

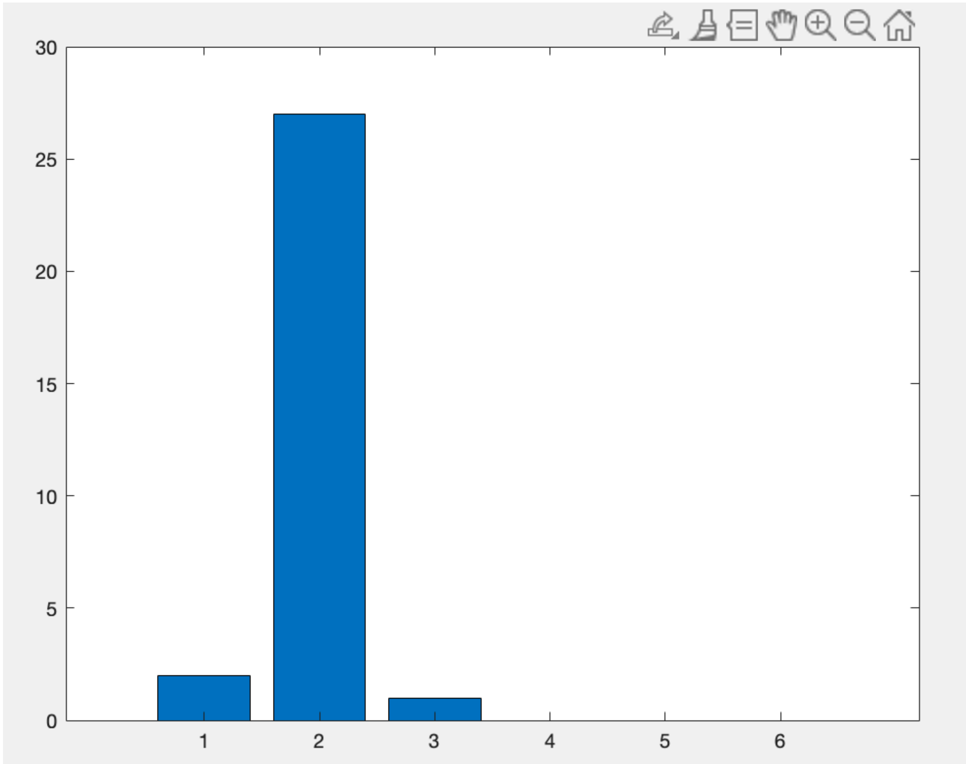
RESULTS

10 SAMPLES



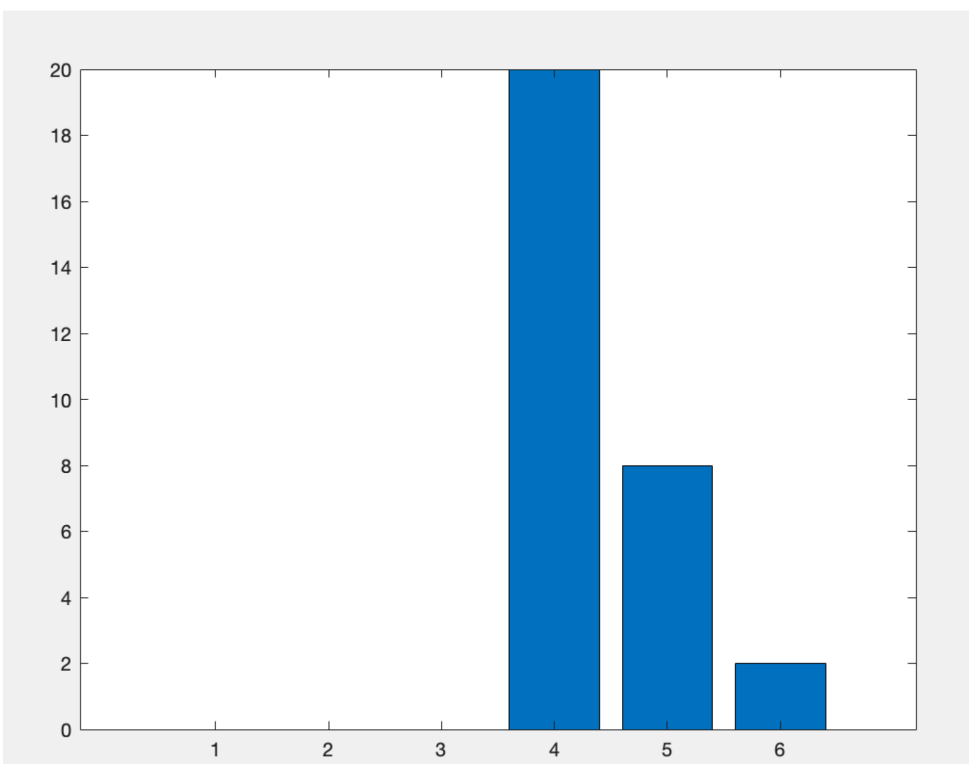
MODEL SELECTED = 1

100 SAMPLES



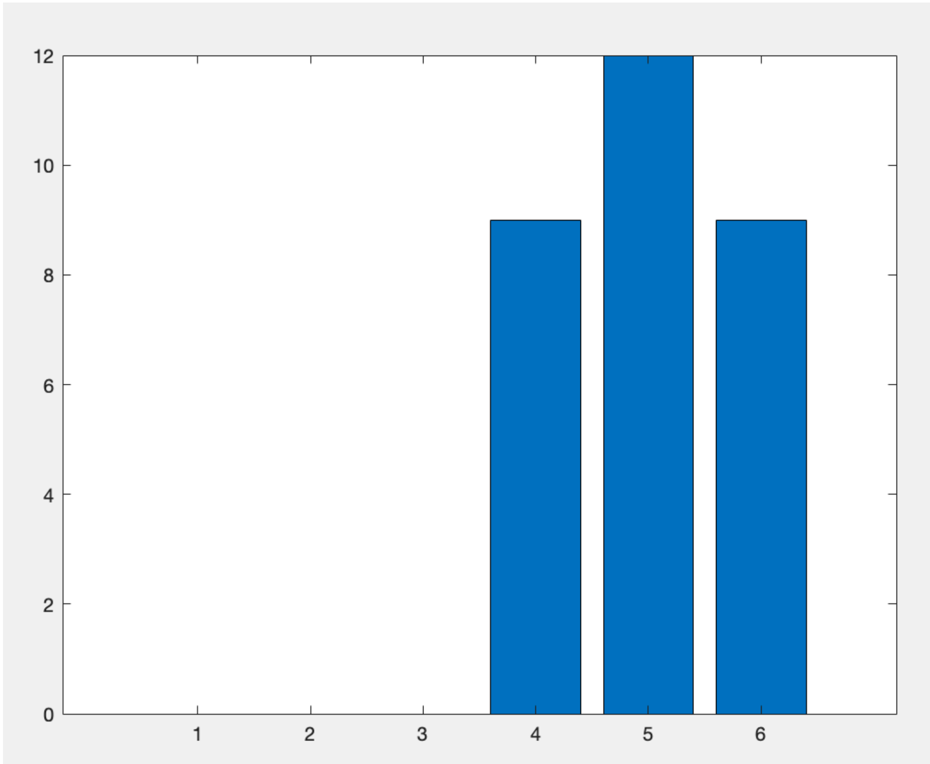
MODEL SELECTED = 2

1000 SAMPLES



MODEL SELECTED = 4

10000 SAMPLES



MODEL SELECTED = 5

OBSERVATIONS

The Gaussian Mixture Model struggles to accurately classify data when there is overlap between clusters, especially with smaller sample sizes, and the convergence of the model can be time-consuming, requiring too many iterations which may result in incorrect classifications. Additionally, the hyperparameters of the model must be carefully tuned to produce effective results, otherwise the model could be suscept to over-fitting.

APPENDIX

CODE FOR QUESTION 1 (MATLAB)

```
clc; clear all; close all;


n = 3;
CL = 4;


priori = [0.25,0.25,0.25,0.25];
mean_ij(1,:) = [0,0,0];
mean_ij(2,:) = [0,0,5];
mean_ij(3,:) = [0,5,0];
mean_ij(4,:) = [5,0,0];
cov_m(:,:,1) = [1,0,-2;0,1,0;-2,0,12];
cov_m(:,:,2) = [6,0,0;0,0.3,0;0,0,0.3];
cov_m(:,:,3) = [1,0,-2;0,0.3,0;-2,0,12];
cov_m(:,:,4) = [6,0,2;0,1,0;2,0,12];


DS_train = [100,200,500,1000,2000,5000];
DS_val = 10000;

%Perceptron Range
PVal = 1:10;

%Learning Rate and Epochs for finding best number of Perceptrons
alpha1 = 0.01;
epochs1 = 100;

%Learning Rate and Epochs for training MLP
alpha2 = 0.01;
epochs2 = 100;

%Generate Test Data
DSval = generate_samples(mean_ij,cov_m,priori,DS_val,n);
Xval = DSval(:,1:n);
mu_n = mean(Xval);
sd_n = std(Xval);
Xval = (Xval - repmat(mu_n, DS_val, 1)) ./ repmat(sd_n, DS_val, 1);
Yval = DSval(:,CL) + 1;

for t = 1:numel(DS_train)
    %Find Optimum Number of Perceptrons
    %Generate Training Data
    DS_size = DS_train(t);
    Dtrain = generate_samples(mean_ij,cov_m,priori,DS_size,n);
    fprintf('MLP MODEL %d (%d training samples)\n',t,DS_size);
    fprintf('-------------------------------------\n');
    % Shuffle the rows of the data randomly
    shf_idx = randperm(size(Dtrain, 1));
    Dtrain_shf = Dtrain(shf_idx, :);

    % Separate the input data from the class labels
    X = Dtrain_shf(:,1:end-1);

    mu_n = mean(X);
    sd_n = std(X);
    X = (X - repmat(mu_n, DS_size, 1)) ./ repmat(sd_n, DS_size, 1);
    Y = Dtrain_shf(:,end);
```

```
Y = Y + 1;

% Convert the class labels to binary vectors
CL = max(Y);
T = zeros(length(Y),CL);
for i = 1:length(Y)
    T(i,Y(i)) = 1;
end

%Splitting the data into 10 folds
k = 10;
f_size = floor(DS_size/k);
Xfolds = zeros(f_size,n,k);
for fold = 1:k
    fold_start = (fold-1)*f_size + 1;
    fold_end = fold*f_size;
    Xfolds(:,:,fold) = X(fold_start:fold_end,:);
end

Tfolds = zeros(f_size,n+1,k);
for fold = 1:k
    fold_start = (fold-1)*f_size + 1;
    fold_end = fold*f_size;
    Tfolds(:,:,fold) = T(fold_start:fold_end,:);
end

% Set up the neural network with ELU activation function and SoftMax output layer
min_ce = 1;
for i = 1:length(PVal)
    inp = size(X,2);
    numH = PVal(i);
    outp = CL;

    % Evaluate the performance of the neural network using 10-fold cross-validation
    fold_ce = zeros(k,1);
    for fold = 1:k
        % Divide the data into training and validation sets
        Xval_fold = squeeze(Xfolds(:,:,fold));
        Tval_fold = squeeze(Tfolds(:,:,fold));

        t_idx = [1:(fold-1)*f_size,fold*f_size+1:DS_size];
        Xtrain_fold = X(t_idx,:);
        Ttrain_fold = T(t_idx,:);
        r_ce = zeros(10,1);
        best_ll = -Inf;
        for j = 1:10
            % Train the neural network on the training set
            wt1 = randn(inp,numH);
            bs1 = randn(numH,1);
            wt2 = randn(numH,outp);
            bs2 = randn(outp,1);
            for epoch = 1:epochs1
                % Forward propagation
                Z1 = Xtrain_fold*wt1 + repmat(bs1',size(Xtrain_fold,1),1);
                A1 = elu(Z1,alpha1);
                Z2 = A1*wt2 + repmat(bs2',size(Xtrain_fold,1),1);
                Y = s_max(Z2);

                % Backward propagation
                dZ2 = Y - Ttrain_fold;
                dwt2 = A1' * dZ2;
```

```
                    dbs2 = sum(dZ2,1)';
                    dA1 = dZ2 * wt2';
                    dZ1 = eluG(Z1) .* dA1;
                    dwt1 = Xtrain_fold' * dZ1;
                    dbs1 = sum(dZ1,1)';

                    % Update the weights and biases of the neural network
                    wt1 = wt1 - alpha1*dwt1;
                    bs1 = bs1 - alpha1*dbs1;
                    wt2 = wt2 - alpha1*dwt2;
                    bs2 = bs2 - alpha1*dbs2;
                    ytrain_fold = zeros(size(Tval_fold,1),1);
                    for s = 1:size(Tval_fold,1)
                        for c = 1:CL
                            if Ttrain_fold(s,c) == 1
                                ytrain_fold(s) = c;
                            end
                        end
                    end
                    train_ll(epoch) = sum(log(Y(ytrain_fold',1)));
                end
                if (train_ll(end) > best_ll)
                    best_ll = train_ll(end);
                    rwt1 = wt1;
                    rwt2 = wt2;
                    rbs1 = bs1;
                    rbs2 = bs2;
                end
            end
            % Evaluate the performance of the trained neural network on the validation
set
            Z1 = Xval_fold*rwt1 + repmat(rbs1',size(Xval_fold,1),1);
            A1 = elu(Z1,alpha1);
            Z2 = A1*rwt2 + repmat(rbs2',size(Xval_fold,1),1);
            Y = s_max(Z2);
            [~,pred] = max(Y,[],2);
            corr = 0;
            for a = 1:numel(pred)
                p = pred(a);
                if (Tval_fold(a,p) == 1)
                    corr = corr + 1;
                end
            end
            fold_ce(fold) = 1 - (corr/numel(pred));
        end

        % Calculate the average error over all folds
        avg_fold_ce = mean(fold_ce);

        % Update the minimum error and the corresponding neural network parameters
        if avg_fold_ce <= min_ce
            min_ce = avg_fold_ce;
            bestwt1 = rwt1;
            bestbs1 = rbs1;
            bestwt2 = rwt2;
            bestbs2 = rbs2;
            bestnumH = numH;
        end
    end
    best_P(t) = bestnumH;
    fprintf('Optimum number of perceptrons = %d\n',bestnumH);
```

```
fprintf('Min Classification Error = %f\n',min_ce);

%Train MLP models using optimum number of perceptrons for each set
Xtrain = Dtrain(:,1:n);
mu_n = mean(Xtrain);
sd_n = std(Xtrain);
Xtrain = (Xtrain - repmat(mu_n, DS_size, 1)) ./ repmat(sd_n, DS_size, 1);
Ytrain = Dtrain(:,n+1) + 1;
Ttrain = zeros(length(Ytrain),CL);
for i = 1:length(Ytrain)
    Ttrain(i,Ytrain(i)) = 1;
end
inp = size(Xtrain,2);
outp = CL;
numH = best_P(t);
best_ll = -Inf;
for j = 1:10
    wt1 = randn(inp,numH);
    bs1 = zeros(numH,1);
    wt2 = randn(numH,outp);
    bs2 = zeros(outp,1);
    train_ll = zeros(epochs2, 1);
    for epoch = 1:epochs2
        % Forward pass
        Z1 = Xtrain*wt1 + repmat(bs1',DS_size,1);
        A1 = sig(Z1);
        Z2 = A1*wt2 + repmat(bs2',DS_size,1);
        Y = s_max(Z2);

        % Compute cross-entropy loss
        loss = -sum(log(Y(Ytrain',1)));

        % Backward pass
        dZ2 = Y - Ttrain;
        dwt2 = A1' * dZ2;
        dbs2 = sum(dZ2,1)';
        dA1 = dZ2 * wt2';
        dZ1 = sigG(Z1) .* dA1;
        dwt1 = Xtrain' * dZ1;
        dbs1 = sum(dZ1,1)';
        wt1 = wt1 - alpha2 * dwt1;
        bs1 = bs1 - alpha2 * dbs1;
        wt2 = wt2 - alpha2 * dwt2;
        bs2 = bs2 - alpha2 * dbs2;
        train_ll(epoch) = sum(log(Y(Ytrain',1)));
    end
    if (train_ll(end) > best_ll)
        best_ll = train_ll(end);
        bestwt1 = wt1;
        bestwt2 = wt2;
        bestbs1 = bs1;
        bestbs2 = bs2;
    end
end

%Test each MLP with the test data
Z1 = Xval*bestwt1 + repmat(bestbs1',DS_val,1);
A1 = sig(Z1);
Z2 = A1*bestwt2 + repmat(bestbs2',DS_val,1);
Y = s_max(Z2);
```

```
    %Calculate Error and Confusion matrix
    [~,pred] = max(Y,[],2);
    conf_m = zeros(CL);
    for i = 1:numel(pred)
        a = Yval(i);
        p = pred(i);
        conf_m(a,p) = conf_m(a,p) + 1;
    end
    error = 1 - (trace(conf_m)/DS_val);
    fprintf('Probability of Error = %f\n',error);
    fprintf('Confusion Matrix =  \n');
    disp(conf_m);
    errors(t) = error;
end

%Theoretical Classifier
%Generate Test Data
DSval = generate_samples(mean_ij,cov_m,priori,DS_val,n);
Xval = DSval(:,1:n);
mu_n = mean(Xval);
sd_n = std(Xval);
Xval = (Xval - repmat(mu_n, DS_val, 1)) ./ repmat(sd_n, DS_val, 1);
Yval = DSval(:,CL) + 1;

%Loss matrix
loss_m = ones(CL) - eye(CL);

%Class-Conditional Probabilities
probX_L = zeros(CL,DS_val);
for i = 1:CL
    probX_L(i,:) = mvnpdf(Xval,mean_ij(i,:),squeeze(cov_m(:,:,i)));
end

%Class Posteriors
probX = priori * probX_L;
Class_Pos = (probX_L .* repmat(priori', 1, DS_val)) ./ repmat(probX, CL, 1);
exp_risk = loss_m * Class_Pos;
[~,dec] = min(exp_risk,[],1);
dec = dec';

%Calculate Risk and Confusion Matrix
avg_exp_risk = sum(min(exp_risk,[],1))/DS_val;
opt_conf_m = zeros(CL);
for i = 1:CL
    for j = 1:CL
        opt_conf_m(i,j) = numel(find((i == Yval) & (j == dec)));
    end
end

% Plot graph to compare theoretical classifier and various MLP models
plot(DS_train, errors, '-', 'MarkerSize', 15);
hold on;
yline(avg_exp_risk, 'r-', 'LineWidth', 1);
xlabel('Number of Samples');
ylabel('Error');
title('Plot to Compare Errors of the trained models');
hold off;

function DS = generate_samples(mean_ij,cov_m,priori,DS_size,n)
    Priori_Cum = cumsum(priori);
    rand = randn(DS_size, 1);
```

```
    CL = zeros(size(rand));
    for i = 1:DS_size
        if rand(i) <= Priori_Cum(1)
            CL(i) = 0;
        elseif rand(i) <= Priori_Cum(2)
            CL(i) = 1;
        elseif rand(i) <= Priori_Cum(3)
            CL(i) = 2;
        else
            CL(i) = 3;
        end
    end
    DS = zeros(DS_size,n);
    for i = 1:DS_size
        if CL(i) == 0
            DS(i,:) = mvnrnd(mean_ij(1,:),squeeze(cov_m(:,:,1)));
        elseif CL(i) == 1
            DS(i,:) = mvnrnd(mean_ij(2,:),squeeze(cov_m(:,:,2)));
        elseif CL(i) == 2
            DS(i,:) = mvnrnd(mean_ij(3,:),squeeze(cov_m(:,:,3)));
        elseif CL(i) == 3
            DS(i,:) = mvnrnd(mean_ij(4,:),squeeze(cov_m(:,:,4)));
        end
    end
    DS = [DS,CL];
end

function y = s_max(x)
    m = max(x,[],2);
    y = exp(x - m) ./ sum(exp(x - m),2);
end

function y = elu(x,alpha)
    y = x .* (x > 0) + alpha * (exp(x) - 1) .* (x <= 0);
end

function y = eluG(x)
    y = ones(size(x));
    y(x < 0) = exp(x(x < 0));
end

function y = sig(x)
    % Sigmoid activation function
    y = 1 ./ (1 + exp(-x));
end

function y = sigG(x)
    % Computes the gradient of the sigmoid function at x
    y = sig(x).*(1-sig(x));
end
```

CODE FOR QUESTION 2 (MATLAB)

```matlab
clc; clear all; close all;


n_features = 2;
n_comp = 4;
priori = [0.2,0.3,0.4,0.1];
comp_mu(1,:) = [0,0];
comp_mu(2,:) = [0,5];
comp_mu(3,:) = [5,0];
comp_mu(4,:) = [5,5];
comp_sd(:,:,1) = [10,4;4,10];
comp_sd(:,:,2) = [8,6;6,8];
comp_sd(:,:,3) = [5,0;0,5];
comp_sd(:,:,4) = [4,1;1,5];
set_size = [10,100,1000,10000];
folds = 10;
g_comp = 6;


select = zeros(numel(set_size),g_comp);
for s = 1:numel(set_size)
    %Generate Data
    n_samples = set_size(s);
    DS = generate_samples(comp_mu,comp_sd,priori,n_samples,n_features,n_comp);
    plot_data(DS);
    n_repeat = 30;


    for rep = 1:n_repeat
        %Shuffle Data
        shf_idx = randperm(n_samples);
        DS = DS(shf_idx, :);

        %Split data into X and Y
        X = DS(:,1:n_features);
        Y = DS(:,n_features+1);

        %Split data into 10 folds
        n_folds = 10;
        f_size = floor(n_samples/n_folds);
        Xfolds = zeros(f_size,n_features,n_folds);
        for f = 1:n_folds
            fold_start = (f-1)*f_size + 1;
            fold_end = f*f_size;
            Xfolds(:,:,f) = X(fold_start:fold_end,:);
        end

        %Perform 10-fold cross validation for each number of components
        max_ll = 0;
        comp_ll = zeros(g_comp,1);
        for g = 1:g_comp
            fold_ll = zeros(n_folds,1);
            for f = 1:n_folds
                %Assign Training and Validation Folds
                Xval_fold = squeeze(Xfolds(:,:,f));
                t_idx = [1:(f-1)*f_size,f*f_size+1:n_samples];
                Xtrain_fold = X(t_idx,:);

                % Estimate GMM parameters using EM algorithm
```

```
                gmFit = fitgmdist(Xtrain_fold, g, 'RegularizationValue', 0.01,
'ProbabilityTolerance',1e-6, 'Options', statset('MaxIter', 1000));

                % Evaluate log-likelihood of validation set
                fold_ll(f) = sum(log(pdf(gmFit,Xval_fold)));
            end
            comp_ll(g) = mean(fold_ll);
        end
        [max_ll,g_select] = max(comp_ll);
        select(s,g_select) = select(s,g_select) + 1;
    end
    fprintf('Set with %d samples completed.\n',n_samples);
    figure
    bar(select(s,:))
end

function DS = generate_samples(comp_mu,comp_sd,priori,n_samples,n,c)

    x = zeros(n, n_samples);
    labels = zeros(1, n_samples);

    u = rand(1, n_samples);
    th = zeros(1, c+1);
    th(1:c) = cumsum(priori);
    th(c+1) = 1;

    for l = 1:c
        indl = find(u <= th(l));
        Nl = length(indl);
        labels(indl) = (l-1)*ones(1, Nl);
        u(indl) = 1.1;
        x(:, indl) = mvnrnd(comp_mu(l, :), squeeze(comp_sd(:,:,l)), Nl)';
    end
    x = x';
    labels = labels' + 1;
    DS = [x,labels];
end

function plot_data(DS)
    figure
    scatter(DS(find(DS(:,3)==1),1),DS(find(DS(:,3)==1),2),'o', 'g')
    hold on
    scatter(DS(find(DS(:,3)==2),1),DS(find(DS(:,3)==2),2),'X', 'b')
    scatter(DS(find(DS(:,3)==3),1),DS(find(DS(:,3)==3),2),'*', 'r')
    scatter(DS(find(DS(:,3)==4),1),DS(find(DS(:,3)==4),2),'+', 'k')

end
```