EECE-5644

INTRO TO MACHINE LEARNING AND PATTERN RECOGNITION

ASSIGNMENT 4

SAI PRASASTH KOUNDINYA GANDRAKOTA
NUID: 002772719

QUESTION 1

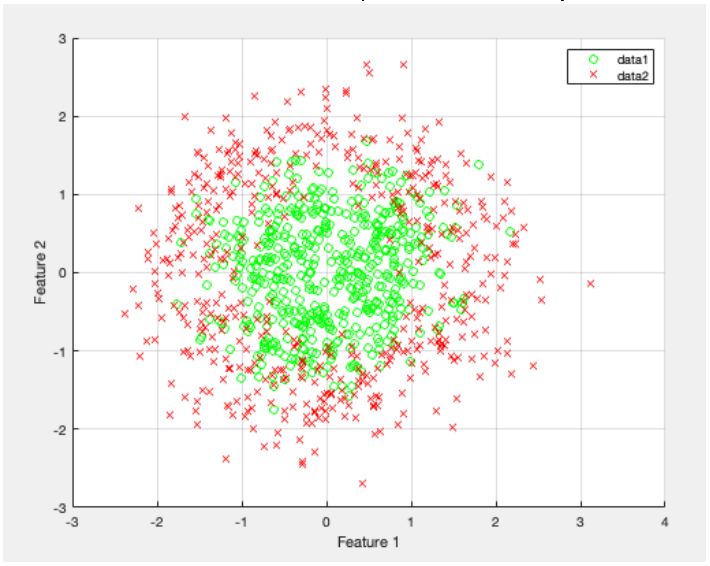
DATA GENERATION
We generate 1000 independent and identically distributed (iid) samples for training and 10000 iid samples for testing. All data for class $l \in \{-1, +1\}$ is generated using the below function:

$x = r_l * [\cos(\theta);\sin(\theta)] + n$

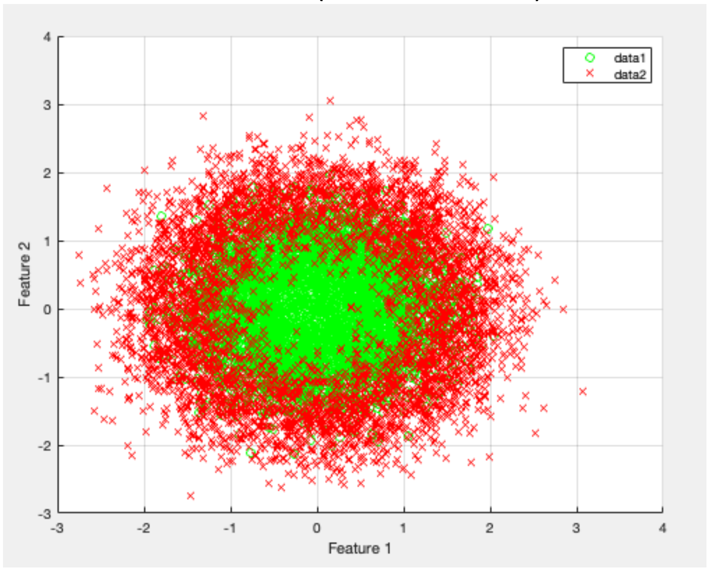
$\theta \sim \text{Uniform}[-\pi, \pi]$
$n \sim N(0, \sigma 2I)$.
$r_{-1} = 2$,
$r_{+1} = 4$
$\sigma = 1$

Since the data is iid, we assume the class priors are 0.5 for each class. In order to facilitate uniform values across the code, we convert the labels into binary form (0,1).

TRAINING DATA (1000 SAMPLES)



TEST DATA (10000 SAMPLES)



MULTI-LAYER PERCEPTRON

ARCHITECTURE

A 2-layer MLP, with one layer of P hidden perceptrons with the Log-Sigmoid 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. We use the Deep Learning Toolbox provided by MATLAB to define the neural network using feedforwadnet.

TRAINING

Having identified the optimum number of perceptrons for each training set, we then train the MLP using the optimal number of perceptrons computed above.

TESTING

Using the trained MLP we then predict the labels of the test data set and estimate the probability of error along with the confusion matrix. We also plot the results of the classification of the test set and show the decision boundary.

SUPPORT VECTOR MACHINE

ARCHITECTURE

To identify the optimal hyperparameters, we perform grid search using 10-fold cross-validation. The hyperparameters are C (box constraint), which controls the balance between misclassification error and maximizing the margin between classes in the optimization function, and sigma (width of the Gaussian kernel). MATLAB's fitcsvm function is used for training the model and predicting the labels.

TRAINING

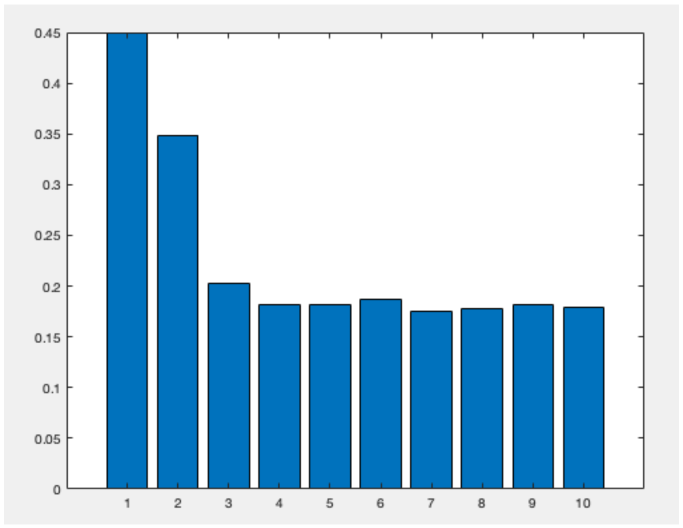
Having identified the optimal hyperparameters, we then train the SVM using those hyperparameters on the training data set.

TESTING

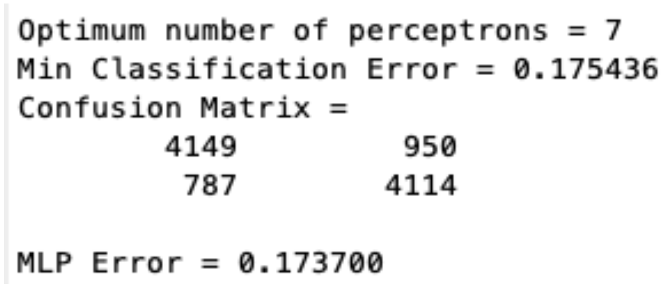
Using the trained SVM we then predict the labels of the test data set and estimate the probability of error along with the confusion matrix. We also plot the results of the classification of the test set and show the decision boundary.

RESULTS
MLP

- The optimum value for number of hidden perceptrons in the hidden layer is found to be 7. As the number of perceptrons increases the error decreases and then stays approximately constant after reaching 7:

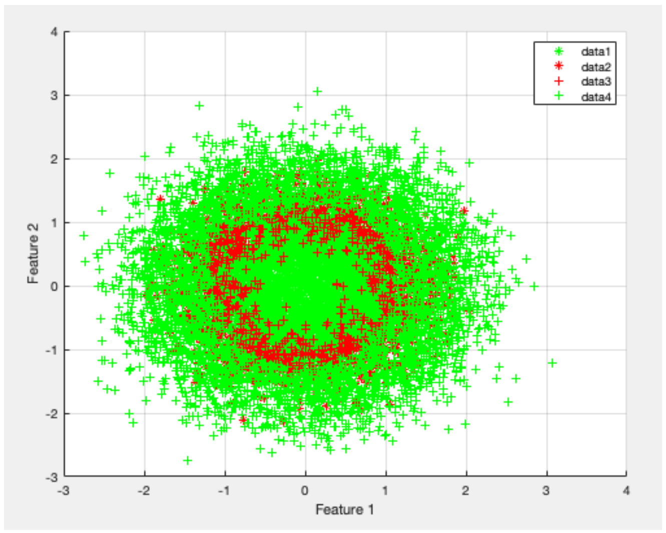


- After training the MLP using 7 perceptrons and testing on the test data, we obtain the following error and confusion matrix:



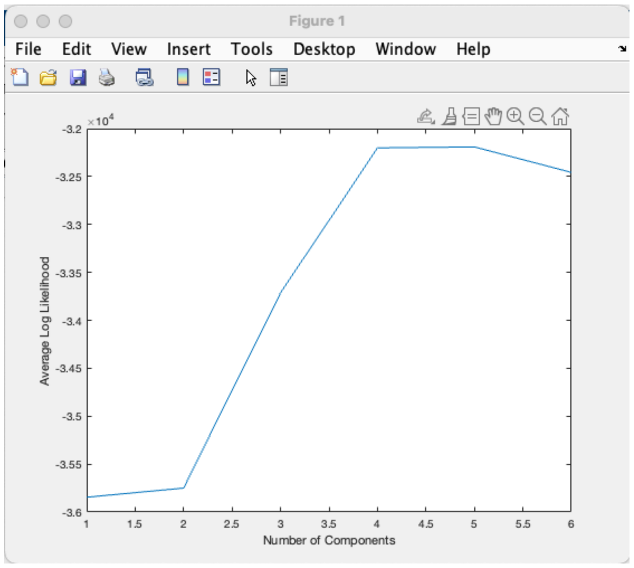
```
Optimum number of perceptrons = 7
Min Classification Error = 0.175436
Confusion Matrix =
        4149         950
         787        4114

MLP Error = 0.173700
```

- The Decision boundary is shown below. We can see the incorrect classifications (red) and correct classifications (green) form a circular shape based on the input data:

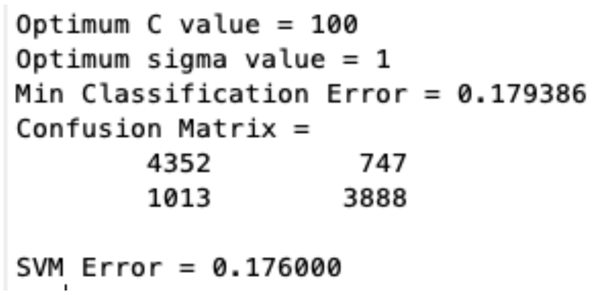


SVM

- The optimum C value and sigma values are computed as 0.1 and 1 respectively. We compute them by finding the maximum log likelihood for all number of components

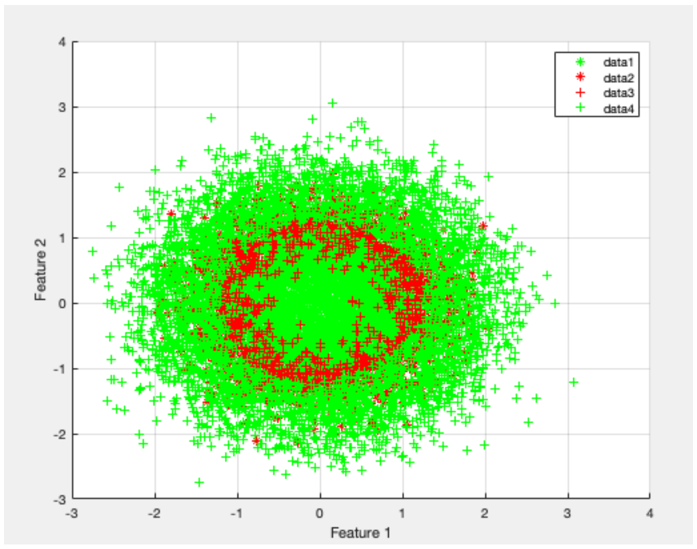


- After training the SVM using the optimal C and sigma values and testing on the test data, we obtain the following error and confusion matrix:



```
Optimum C value = 100
Optimum sigma value = 1
Min Classification Error = 0.179386
Confusion Matrix =
        4352         747
        1013        3888

SVM Error = 0.176000
```

- The Decision boundary is shown below. We can see the incorrect classifications (red) and correct classifications (green) form a circular shape based on the input data:

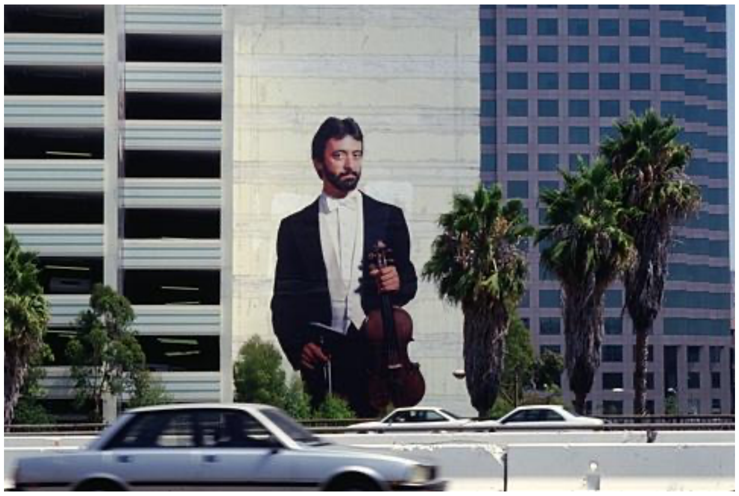


- **Both errors obtained from the final MLP and SVM classifiers are almost similar.**

QUESTION 2

INPUT IMAGE

The input image taken was of size 321 x 481 pixels and consisted of three color channels (R,G,B). The image was downsampled to 60 percent of its size to reduce the number of iterations in the code. A raw feature vector was then created which contained the row number, column, number, R value, G value and B value for every pixel in the input image. This vector is then normalized to a uniform range between 0 and 1.



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 maximum number of 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

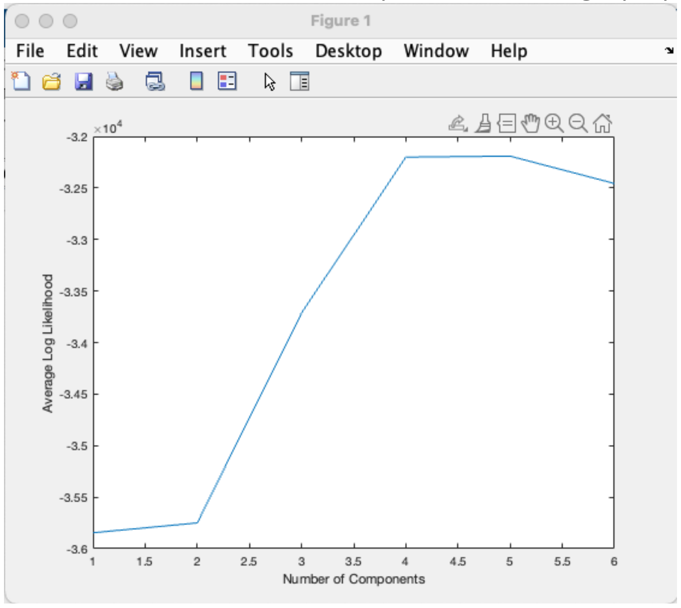
Each GMM order with Gaussian components ranging from 1 to 6 is cross validated by splitting the data into 10 folds, taking each fold as a validation fold and the remaining as the training folds, we fit the GMM distribution onto the training data and then calculate the log likelihood of the validation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

MODEL ORDER SELECTION

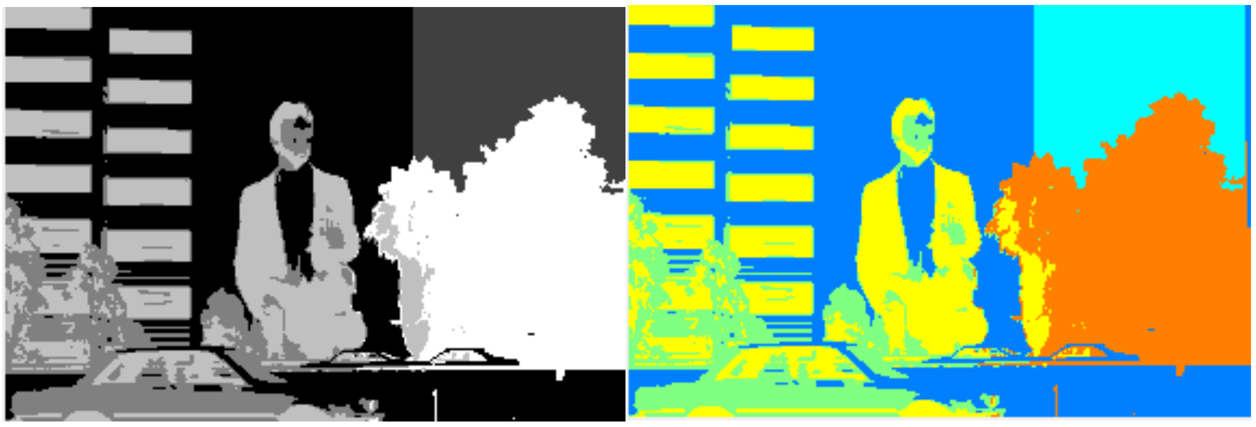
After computing the scores for different GMM models, the one with the highest score is selected as the best model. This best model is then used to calculate the probabilities of each pixel belonging to different color clusters using GMM. To assign pixels to a specific cluster, the Maximum A Posteriori (MAP) classification rule is applied, which uses the computed posterior probabilities.

RESULTS

- The best GMM model selected is the model with4 components as the graph peaks at that value



- The segmented image clearly shows the outlines of the shapes from the original image. Certain shapes like the windows on the buildings in the background are not clearly distinguished. However, these could also be segmented given we use more iterations, reduce the noise in the input image, or use a larger input image.



OBSERVATIONS

The Gaussian Mixture Model struggles to accurately segment the image 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)

```
%Q1
clc; clear; close all;

n_train = 1000;
n_test = 10000;
r_minus = 2;
r_plus = 4;

n_classes = 2;
n_features = 2;

%Generate Samples
[Xtrain,Ytrain] = generate_samples(n_train,r_minus,r_plus,n_features);
[Xtest,Ytest] = generate_samples(n_test,r_minus,r_plus,n_features);

%Plot Input Data
plot_data(Xtrain,Ytrain)
plot_data(Xtest,Ytest)

%MLP
n_perceptrons = 10;
num_folds = 10;

%Create output matrix
outputMatrix = zeros(n_classes, n_train);
for i = 1:n_classes
    outputMatrix(i,:) = (Ytrain == i-1);
end

%Create folds
f_size = n_train/num_folds;
fold_i = [1:f_size:n_train,n_train];

%Find optimal number of perceptrons
min_ce = 1;
for i = 1:n_perceptrons

    %10 Fold cross validation
    for j = 1:num_folds
        val_indices = fold_i(j):fold_i(j+1);
        train_indices = setdiff(1:n_train,val_indices);

        %Train MLP
        net = feedforwardnet(i);
        net.trainParam.epochs = 100;
        net.trainParam.lr = 0.01;
        net.layers{1}.transferFcn = 'logsig';
        net.layers{end}.transferFcn = 'softmax';
        net.trainParam.showWindow = false;
        [net,~] = train(net, Xtrain(:,train_indices), outputMatrix(:,train_indices));

        %Validate MLP
        yVal = net(Xtrain(:,val_indices));
        [~, pred] = max(yVal);
        pred = pred - 1;
        targ = Ytrain(val_indices);
        corr = 0;
        for n = 1:numel(pred)
```

```
            if pred(n) == targ(n)
                corr = corr + 1;
            end
        end
        fold_ce(j) = 1 - (corr/numel(pred));
    end
    avg_fold_ce(i) = mean(fold_ce);

    % Update the minimum error and the corresponding neural network parameters
    if avg_fold_ce(i) <= min_ce
        min_ce = avg_fold_ce(i);
        best_n_hidden = i;
    end
end
disp([avg_fold_ce;1:10]);
fprintf('Optimum number of perceptrons = %d\n',best_n_hidden);
fprintf('Min Classification Error = %f\n',min_ce);

%Train MLP using optimum number of perceptrons
net = feedforwardnet(i);
net.trainParam.epochs = 100;
net.trainParam.lr = 0.01;
net.layers{1}.transferFcn = 'logsig';
net.layers{end}.transferFcn = 'softmax';
net.trainParam.showWindow = false;
[net,~] = train(net, Xtrain, outputMatrix);

%Validate MLP
yVal = net(Xtest);
[~, pred] = max(yVal);
pred = pred - 1;
targ = Ytest;
plot_output(Xtest,Ytest,pred)
conf_M = zeros(n_classes);
for n = 1:numel(pred)
    p = pred(n) + 1;
    t = targ(n) + 1;
    conf_M(t,p) = conf_M(t,p) + 1;
end
error = 1 - (trace(conf_M)/n_test);
fprintf('Confusion Matrix =  \n');
disp(conf_M);
fprintf('MLP Error = %f\n',error);

%SVM
Xtrain = Xtrain';
Ytrain = Ytrain';
Xtest = Xtest';
Ytest = Ytest';
% Set hyperparameters to try
C_vals = [0.01,0.1,1,10,100];
sigma_vals = [0.001,0.01,0.1,1];

% Split data into 10 folds
%Create folds
num_folds = 10;
f_size = n_train/num_folds;
fold_i = [1:f_size:n_train,n_train];

% Perform grid search using 10-fold cross-validation
best_err = 1;
```

```matlab
for C = C_vals
    for sigma = sigma_vals
        err = zeros(num_folds,1);
        for i = 1:num_folds
            % Split data into training and validation sets
            val_indices = fold_i(i):fold_i(i+1);
            train_indices = setdiff(1:n_train,val_indices);
            X_train_fold = Xtrain(train_indices,:);
            Y_train_fold = Ytrain(train_indices);
            X_val_fold = Xtrain(val_indices,:);
            Y_val_fold = Ytrain(val_indices);

            % Train SVM classifier using training set
            svm_model = fitcsvm(X_train_fold, Y_train_fold, 'BoxConstraint', C,
'KernelFunction', 'gaussian', 'KernelScale', sigma);

            % Predict labels for validation set and compute error
            pred = predict(svm_model, X_val_fold);
            corr = 0;
            for n = 1:numel(pred)
                if pred(n) == Y_val_fold(n)
                    corr = corr + 1;
                end
            end
            err(i) = 1 - (corr/numel(val_indices));
        end

        % Compute average error over all folds
        avg_err = mean(err);

        % Update best hyperparameters if average error is lower
        if avg_err < best_err
            best_C = C;
            best_sigma = sigma;
            best_err = avg_err;
        end
    end
end

fprintf('Optimum C value = %d\n',best_C);
fprintf('Optimum sigma value = %d\n',best_sigma);
fprintf('Min Classification Error = %f\n',best_err);

% Train SVM classifier using entire training set with best hyperparameters
svm_model = fitcsvm(Xtrain, Ytrain, 'BoxConstraint', best_C, 'KernelFunction',
'gaussian', 'KernelScale', best_sigma);

% Test SVM classifier on testing set and compute probability of error
pred = predict(svm_model, Xtest);
plot_output(Xtest',Ytest',pred')
targ = Ytest + 1;
conf_M = zeros(n_classes);
for n = 1:numel(pred)
    p = pred(n) + 1;
    t = targ(n);
    conf_M(t,p) = conf_M(t,p) + 1;
end
error = 1 - (trace(conf_M)/n_test);
fprintf('Confusion Matrix =  \n');
disp(conf_M);
fprintf('SVM Error = %f\n',error);
```

```
function [X,Y] = generate_samples(n_samples,r_minus,r_plus,n_features)
    % Generate data
    X = zeros(n_samples, n_features);
    priori = [0.5,0.5];
    Y = zeros(1, n_samples);
    for i = 1:n_samples
        if rand >= priori(2)
            Y(i) = 0;
        else
            Y(i) = 1;
        end
    end
    sd = 1;
    ind0 = find(Y == 0);
    ind1 = find(Y == 1);
    N0 = numel(ind0);
    N1 = numel(ind1);
    theta0 = 2*pi*randn(N0, 1);
    theta1 = 2*pi*randn(N1, 1);
    x0 = sd^2*randn(N0,n_features) + r_minus.*[cos(theta0), sin(theta0)];
    x1 = sd^2*randn(N1,n_features) + r_plus.*[cos(theta1), sin(theta1)];
    X(ind0,:) = x0;
    X(ind1,:) = x1;
    mu_n = mean(X);
    sd_n = std(X);
    X = (X - repmat(mu_n,n_samples,1)) ./ repmat(sd_n,n_samples,1);
    X = X';
end

function plot_data(X,Y)
    figure
    scatter(X(1,find(Y==0)),X(2,find(Y==0)),'o', 'g')
    hold on
    scatter(X(1,find(Y==1)),X(2,find(Y==1)),'X', 'r')
    xlabel('Feature 1');
    ylabel('Feature 2');
    grid on;
    legend show;
end

function plot_output(X,Y,pred)
    X1 = X(1,:);
    X2 = X(2,:);
    figure
    scatter(X1(find(Y==0 & pred==0)), X2(find(Y==0 & pred==0)),'*','g');
    hold on;
    scatter(X1(find(Y==0 & pred==1)), X2(find(Y==0 & pred==1)),'*','r');
    scatter(X1(find(Y==1 & pred==0)), X2(find(Y==1 & pred==0)),'+','r');
    scatter(X1(find(Y==1 & pred==1)), X2(find(Y==1 & pred==1)),'+','g');
    xlabel('Feature 1');
    ylabel('Feature 2');
    grid on;
    legend show;
end
```

CODE FOR QUESTION 2 (MATLAB)

```
%Q2
clc; clear; close all;

% Read the image from the specified path
path = '/Users/prasasth/Desktop/';
file_name = '119082.jpeg';
image = imread(fullfile(path,file_name));

disp(size(image));

% Downsample the image with specified scale
p_scale = 60; % percent of original size
w = int32(size(image,2) * (p_scale / 100));
h = int32(size(image,1) * (p_scale / 100));
dim = [h,w];

% Resizing the input image
img = imresize(image,dim,'method','bilinear');

% Intialize parameters of the image
rows = size(img,1);
columns = size(img,2);
n_channels = size(img,3);
n_pixels = rows * columns;
n_features = 5;

% Create raw feature vector
raw_FV = zeros(n_pixels,n_features);
p = 1;
for r = 1:rows
    for c = 1:columns
        raw_FV(p,1) = r;
        raw_FV(p,2) = c;
        for ch = 1:n_channels
            raw_FV(p,ch + 2) = img(r,c,ch);
        end
        p = p + 1;
    end
end

% Normalize the feature values
raw_FV = normalize(raw_FV);

%Split data into 10 folds
n_folds = 10;
f_size = floor(n_pixels/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) = raw_FV(fold_start:fold_end,:);
end
avg_LL = [];
n_comp = 1:6;

% Test model orders ranging from 1 to 6
for n = n_comp
    % Initialize a list to store the log-likelihoods for each fold
    comp_LL = zeros(n_folds,1);
    for fold = 1:n_folds
```

```
        % Get training and validation sets
        Xval_fold = squeeze(Xfolds(:,:,f));
        t_idx = [1:(f-1)*f_size,f*f_size+1:n_pixels];
        Xtrain_fold = raw_FV(t_idx,:);

        % Fit the GMM using the EM algorithm
        gmdist = 
fitgmdist(Xtrain_fold,n,'RegularizationValue',0.01,'ProbabilityTolerance',1e-
6,'Options',statset('MaxIter',3000));

        % Calculate the log-likelihood of the validation set
        LL = sum(log(pdf(gmdist,Xval_fold)),'all');

        % Append the log-likelihood to the list for this fold
        comp_LL(fold) = LL;
    end

    % Calculate the average log-likelihood across all K folds
    avg_comp_LL = mean(comp_LL);
    avg_LL(n) = avg_comp_LL;
end


% Plot Log Likelihood Score with respect to number of components in each model
plot(n_comp,avg_LL);
xlabel('Number of Components');
ylabel('Average Log Likelihood');

% Final Model Fitting
[~,max_ll] = max(avg_LL);
best_n_comp = n_comp(max_ll);
best_gmdist_model = 
fitgmdist(raw_FV,best_n_comp,'RegularizationValue',0.01,'ProbabilityTolerance',1e-
6,'Options',statset('MaxIter',3000));

% Compute class post using model weights and conditional probabilties
post = zeros(best_n_comp,n_pixels);
for i = 1:best_n_comp
    PDF = mvnpdf(raw_FV,best_gmdist_model.mu(i,:),best_gmdist_model.Sigma(:,:,i));
    post(i,:) = (best_gmdist_model.PComponents(i) * PDF)';
end

% Decide label for each pixel with maximum posterior value
[~,seg_image] = max(post);

% Reshape seg_image to the shape of the input image
seg_image = reshape(seg_image,[columns,rows]);
seg_image = seg_image';

% Display the segmented image
figure;
imshow(seg_image,[]);
seg_image = label2rgb(seg_image);
figure;
imshow(seg_image,[]);
```

CITATIONS

1) https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/segbench/BSDS300/html/dataset/images.html