EECE5644: Assignment #4

Due on April 13, 2020

Professor Deniz Erdogmus

Anja Deric

GitHub Repo: https://github.com/AnjaDeric/MachineLearning

Problem 1

For the first problem on this exam, a provided function was used to generate 2 data sets- a training data set with 1000 samples, and a test/validation data set with 10000 samples. Pictured below in Figure 1 is the training data set and in Figure 2 is the validation data set.

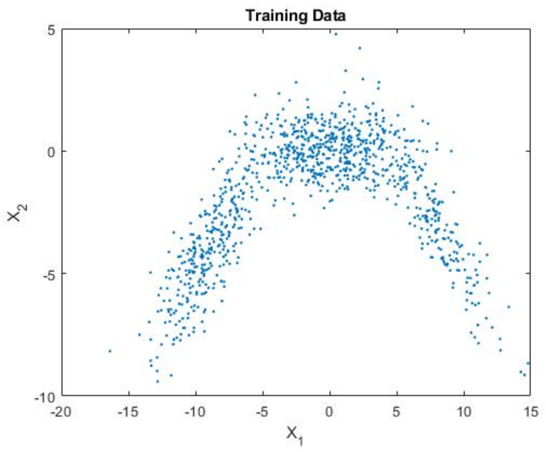


Figure 1: Question 1 Training Samples

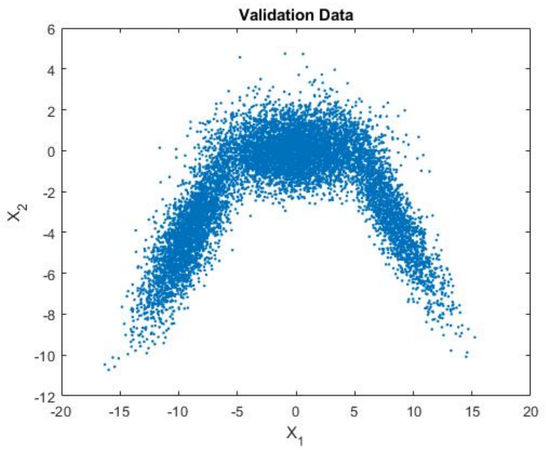
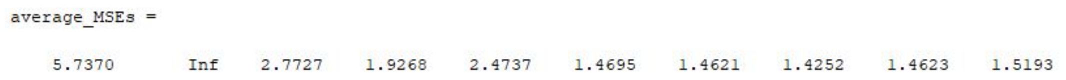


Figure 2: Question 1 Test Samples

After the data was generated, 10-fold cross-validation was used to train and test a single hidden layer MLP to approximate output values given 1-dimensional input values from the data generated above. The softplus activation function was used, and mean-squared-error (MSE) was used to track performance of each trained network. The number of perceptrons was varied from 1 to 10, and Figure 3 on the following page shows the average MSE across 10 trials (with different train and test samples pulled from the original training set) for different number of perceptrons, where the the column number indicates the number of perceptrons.



```
average_MSEs =

    5.7370       Inf    2.7727    1.9268    2.4737    1.4695    1.4621    1.4252    1.4623    1.5193
```

Figure 3: Average MSE for Varying Perceptron Numbers

As can be seen from the data, the lowest average MSE was observed with 8 perceptrons, resulting in an 8-perceptron model being selected as the final pick. With this model structure (8 perceptrons, softplus activation function), one last model was trained using the full training set of 1000 samples. Following that, the trained MLP was applied to the 10000 test samples, and MSE of 1.5283 was achieved as the final performance of the network. Figure 4 below shows the estimated output for each given input, as well as the actual output from the samples used to test the neural net.

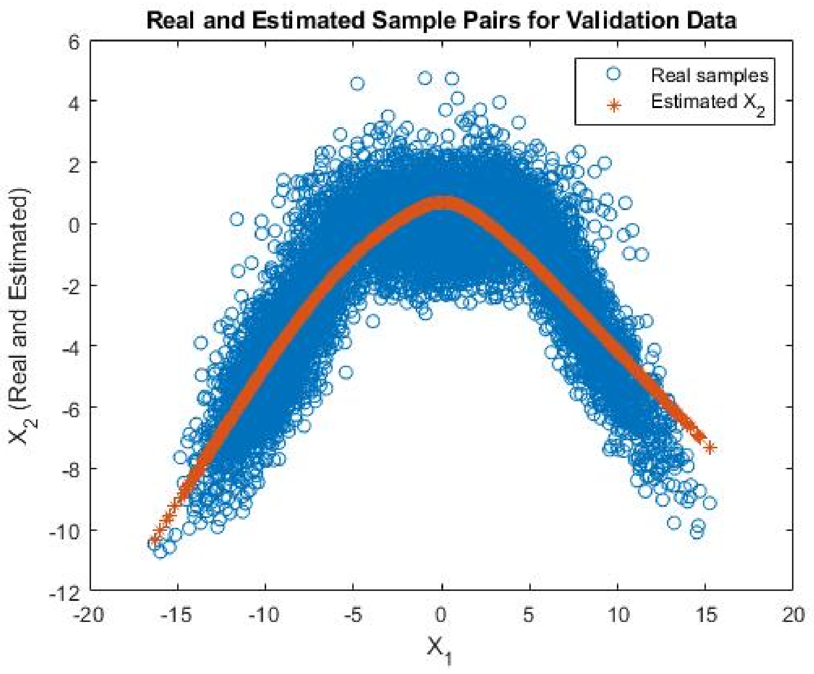


Figure 4: Final Output Estimates

All Matlab code for question 1 can be found in Appendix A of the report.

Problem 2

For the second problem on this exam, a provided function was used to generate 2 data sets- a training data set with 1000 samples, and a test/validation data set with 10000 samples. Pictured below in Figure 5 is the training data set and in Figure 6 is the validation data set.

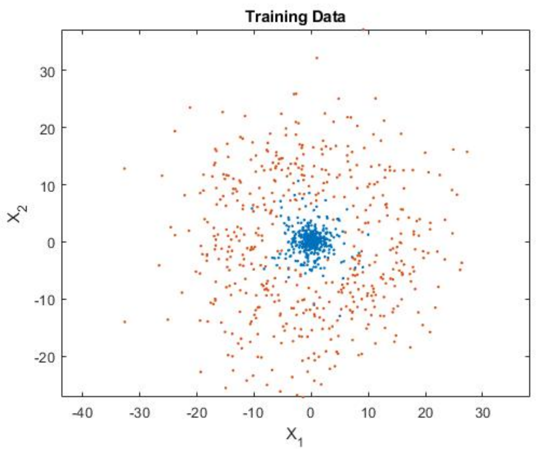


Figure 5: Question 2 Training Samples

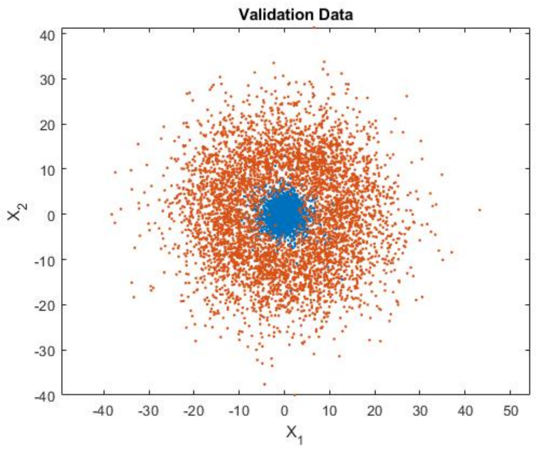
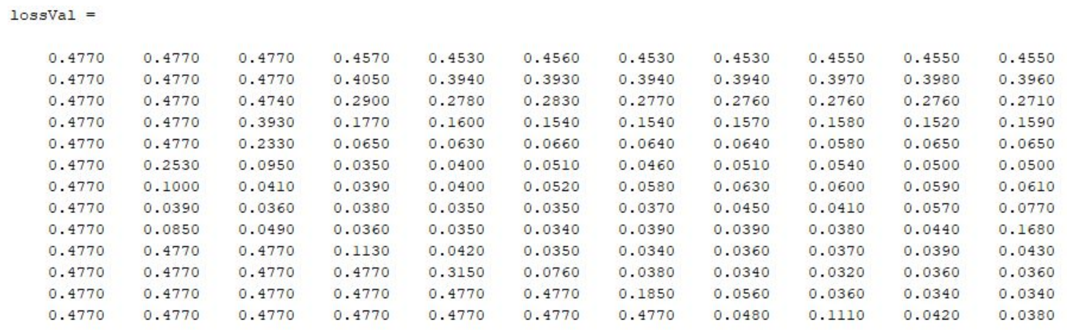


Figure 6: Question 2 Test Samples

In the problem, the training set of 2-dimensional data generated in the previous step was first used to train and evaluate a support vector machine (SVM) classifier with a spherically symmetric Gaussian/RBF kernel. 10-fold cross validation was used to test out several combinations of box constraint hyperparameters C and Gaussian kernel widths σ. In total, 11 C values equally spaced between 10^{-3} and 10^{7} were tested in combination with 13 different σ values, which were equally spaced between 10^{-2} and 10^{3}. For each combination of the parameters, average loss values were computed after 10-fold cross-validation was performed (using the built-in crossval and kfoldLoss Matlab SVM functions respectively). Figure 7 on the next page shows the loss values fo reach combination of the parameters, with rows representing the σ index values, and columns representing the hyperparameter C index values.



```
lossVal =

    0.4770    0.4770    0.4770    0.4570    0.4530    0.4560    0.4530    0.4530    0.4550    0.4550    0.4550
    0.4770    0.4770    0.4770    0.4050    0.3940    0.3930    0.3940    0.3940    0.3970    0.3980    0.3960
    0.4770    0.4770    0.4740    0.2900    0.2780    0.2830    0.2770    0.2760    0.2760    0.2760    0.2710
    0.4770    0.4770    0.3930    0.1770    0.1600    0.1540    0.1540    0.1570    0.1580    0.1520    0.1590
    0.4770    0.4770    0.2330    0.0650    0.0630    0.0660    0.0640    0.0640    0.0580    0.0650    0.0650
    0.4770    0.2530    0.0950    0.0350    0.0400    0.0510    0.0460    0.0510    0.0540    0.0500    0.0500
    0.4770    0.1000    0.0410    0.0390    0.0400    0.0520    0.0580    0.0630    0.0600    0.0590    0.0610
    0.4770    0.0390    0.0360    0.0380    0.0350    0.0350    0.0370    0.0450    0.0410    0.0570    0.0770
    0.4770    0.0850    0.0490    0.0360    0.0350    0.0340    0.0390    0.0390    0.0380    0.0440    0.1680
    0.4770    0.4770    0.4770    0.1130    0.0420    0.0350    0.0340    0.0360    0.0370    0.0390    0.0430
    0.4770    0.4770    0.4770    0.4770    0.3150    0.0760    0.0380    0.0340    0.0320    0.0360    0.0360
    0.4770    0.4770    0.4770    0.4770    0.4770    0.4770    0.1850    0.0560    0.0360    0.0340    0.0340
    0.4770    0.4770    0.4770    0.4770    0.4770    0.4770    0.4770    0.0480    0.1110    0.0420    0.0380
```

Figure 7: Loss Values for Parameter Combinations

As can be observed from the data, the best combination consisted of hyperparameter at index value of 9 (corresponding to C value of 100000), and σ at index value of 11 (correcponding to Gaussian kernel width of 146.8).

After the best combination of parameters was found, one final SVM classifier was trained based on these parameters and used to predict the label of the test/validation set with 10000 samples. After completing this process, the accuracy of the final model was calculated to be 96.95%. Figure 8 below displays the test samples and their classification, with green points representing the correctly classified samples, and the red points representing the incorrectly classified samples.

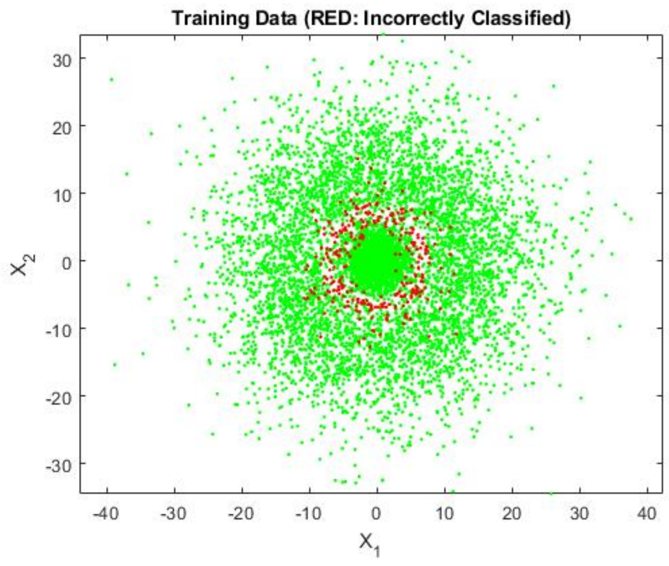


Figure 8: Final Classification Performance

All Matlab code for question 2 can be found in Appendix B of the report.

Problem 3

In this problem, GMM-based clustering was used to segment 2 different images- an airplane image and a bird image. In order to accomplish this, data from the images first had to be normalized across all dimensions. A 5-dimensional feature vector was generated for each image, with the dimensions being row index, column index, and R, G, and B values. Each individual feature was mapped into a [0,1] range in this pre-processing stage.

After creating the normalized feature vector, each image was first fit to a 2-component GM Model using the built in fitgmdist function in Matlab. Following that, the cluster function was used to assign labels to each pixel of the image. After reshaping the image into its original size, the 2-component image was displayed (results can be seein in Figure 9).

Then, for each image, 10-fold cross-validation was used to fit the image to various model orders (from 1 to 6), and the average log-likelihood was used to establish which model order is the best fit for each image. Once the best fit was decided, one final GMM was fit to the image and the cluster function was used again to label each pixel with the cluster number. Once again, the image was displayed next to the original and is also shown in Figure 9.

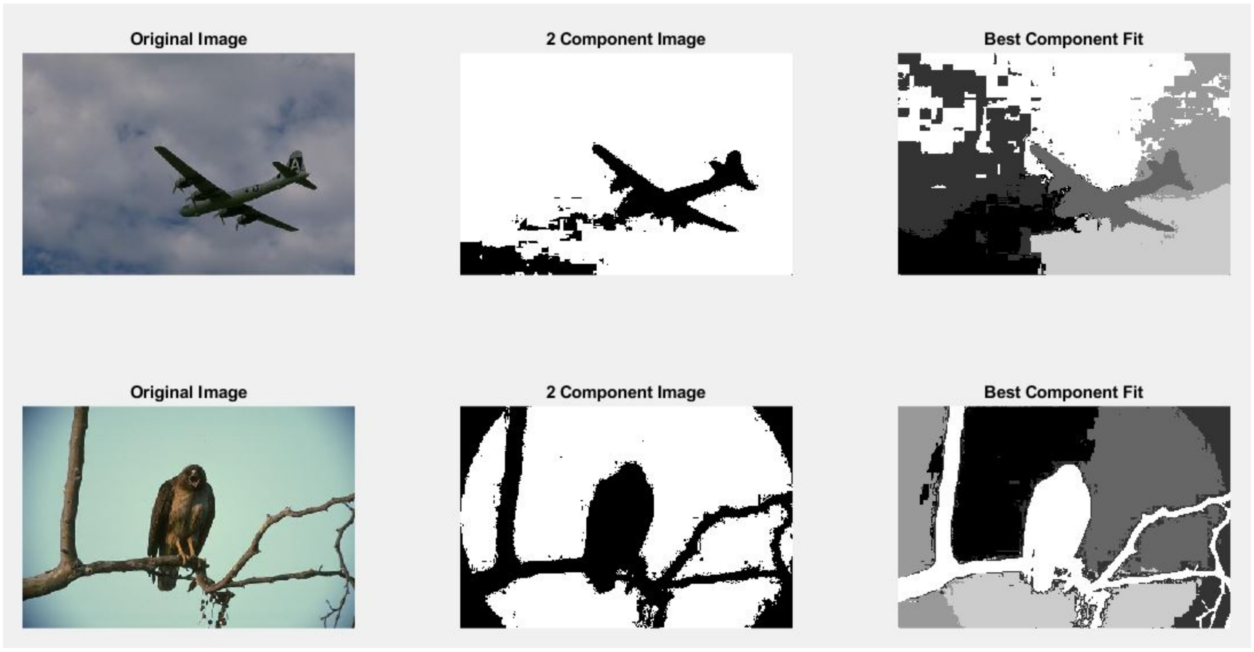
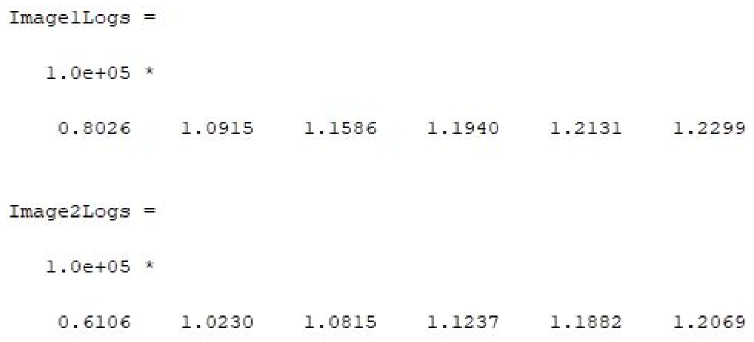


Figure 9: Question 3 Image Segmentation

Problem 3 continued on next page. . .

As can be observed, for both images, the algorithm picked 6 clusters as the best fit. Figure 10 below shows the average log likelihood for each model order and for each image. Each colum number represents the model order, and as expected, the last column (model order 6), has the greatest log likelihood of the component values tested.



```
Image1Logs =

   1.0e+05 *

    0.8026    1.0915    1.1586    1.1940    1.2131    1.2299


Image2Logs =

   1.0e+05 *

    0.6106    1.0230    1.0815    1.1237    1.1882    1.2069
```

Figure 10: Question 3 Model Order Selection

All code for this portion of the assignment can be found in Appendix C.

Appendix A: Question 1 Code

```
%% Take Home Exam 4: Question 1
% Anja Deric | April 13, 2020

% Clear all variables and generate data
clc; clear;
all_train_data = generateData(1000, 'Training Data');
all_test_data = generateData(10000, 'Validation Data');

%% Neural Network Training

% Prepare 10-fold cross-calidations sets and perceptron values
kfold_split = cvpartition(length(all_train_data),'KFold',10);
max_perceptrons = 10; MSE = zeros(kfold_split.NumTestSets,max_perceptrons);

for perceptrons = 1:max_perceptrons
    for test_set = 1:kfold_split.NumTestSets
        [perceptrons test_set]
        % Get train and test data and labels for each test set
        train_index = kfold_split.training(test_set);
        test_index = kfold_split.test(test_set);
        train_data = all_train_data(:,find(train_index));
        test_data = all_train_data(:,find(test_index));

        % Train NN and get MSE value
        MSE(test_set,perceptrons) = mleMLPwAWGN(train_data,test_data,...
            perceptrons,0);
    end
end

% Pick number of perceptrons with lowest average MSE
average_MSEs = mean(MSE,1);
[~, best_perceptron] = min(average_MSEs);

% Train and test final network
final_MSE = mleMLPwAWGN(all_train_data,all_test_data,perceptrons,1);

%% Functions

function MSE = mleMLPwAWGN(train_data, test_data, perceptrons,final)
    % Maximum likelihood training of a 2-layer MLP assuming AWGN

    % Determine/specify sizes of parameter matrices/vectors
    nX = 1;     % input dimensions
    nY = 1; % number of classes
    sizeParams = [nX;perceptrons;nY];

    % True input and output data for training
    X = train_data(1,:); Y = train_data(2,:);

    % Initialize model parameters
    params.A = 0.3*rand(perceptrons,nX);
    params.b = 0.3*rand(perceptrons,1);
    params.C = 0.3*rand(nY,perceptrons);
    params.d = mean(Y,2);
    vecParamsInit = [params.A(:);params.b;params.C(:);params.d];

    % Optimize model using fminsearch
    vecParams = fminsearch(@(vecParams)(objectiveFunction(X,Y,...
        sizeParams,vecParams)),vecParamsInit);

    % Extract best parameters for the model
    params.A = reshape(vecParams(1:nX*perceptrons),perceptrons,nX);
    params.b = vecParams(nX*perceptrons+1:(nX+1)*perceptrons);
    params.C = reshape(vecParams((nX+1)*perceptrons+1:(nX+1+nY)*perceptrons),
       nY,perceptrons);
    params.d = vecParams((nX+1+nY)*perceptrons+1:(nX+1+nY)*perceptrons+nY);

    % Apply trained MLP to test data
    H = mlpModel(test_data(1,:),params);

    % Calculate mean-squared-error
    MSE = (1/length(H))*sum((H-test_data(2,:)).^2);

    % If evaluating final model, plot results
    if final == 1
        % Plot true and estimated data pairs
        figure;
        plot(test_data(1,:),test_data(2,:),'o',test_data(1,:),H,'*');
        xlabel('X_1'); ylabel('X_2 (Real and Estimated)');
        legend('Real samples','Estimated X_2');
        title('Real and Estimated Sample Pairs for Validation Data');
    end
end

function objFncValue = objectiveFunction(X,Y,sizeParams,vecParams)
    % Function to be optimized by neaural net
    N = size(X,2);
    nX = sizeParams(1);
    nPerceptrons = sizeParams(2);
    nY = sizeParams(3);
    params.A = reshape(vecParams(1:nX*nPerceptrons),nPerceptrons,nX);
    params.b = vecParams(nX*nPerceptrons+1:(nX+1)*nPerceptrons);
    params.C = reshape(vecParams((nX+1)*nPerceptrons+1:(nX+1+nY)*nPerceptrons)
        ,nY,nPerceptrons);
    params.d = vecParams((nX+1+nY)*nPerceptrons+1:(nX+1+nY)*nPerceptrons+nY);
    H = mlpModel(X,params); % neural net model
    objFncValue = sum(sum((Y-H).*(Y-H),1),2)/N;
    %objFncValue = sum(-sum(Y.*log(H),1),2)/N;
end

function H = mlpModel(X,params)
    % Neutal Network Model
    N = size(X,2);                          % number of samples
    nY = size(params.d);                    % number of outputs
    U = params.A*X + repmat(params.b,1,N);  % u = Ax + b, x \in R^nX, b,u \in
       R^nPerceptrons, A \in R^{nP-by-nX}
    Z = activationFunction(U);      % z \in R^nP, using nP instead of
       nPerceptons
    V = params.C*Z + repmat(params.d,1,N);  % v = Cz + d, d,v \in R^nY, C \in
       R^{nY-by-nP}
    H = V; % linear output layer
end

function out = activationFunction(in)
    % Softplus activation function
    out = log(1+exp(in));
end

function x = generateData(N,plot_title)
    % Generate and plot dataset with N samples

    % Data mean, variance, priors
    m(:,1) = [-9;-4]; Sigma(:,:,1) = 4*[1,0.8;0.8,1];
    m(:,2) = [0;0]; Sigma(:,:,2) = 3*[3,0;0,0.3];
    m(:,3) = [8;-3]; Sigma(:,:,3) = 5*[1,-0.9;-0.9,1];
    componentPriors = [0.3,0.5,0.2]; thr = [0,cumsum(componentPriors)];

    % Generate Data
    u = rand(1,N); L = zeros(1,N); x = zeros(2,N);
    for l = 1:3
        indices = find(thr(l)<=u & u<thr(l+1)); % if u happens to be precisely
            1, that sample will get omitted - needs to be fixed
        L(1,indices) = l*ones(1,length(indices));
        x(:,indices) = mvnrnd(m(:,l),Sigma(:,:,l),length(indices))';
   end

    % Plot data
    figure; plot(x(1,:),x(2,:),'.');
    xlabel('X_1'); ylabel('X_2');
    title(plot_title);
end
```

Appendix B: Question 2 Code

```
%% Take Home Exam 4: Question 2
% Anja Deric | April 13, 2020

% Clear all variables and generate new training and testing data
clc; clear;
[train_data,train_labels] =  generateMultiringDataset(2,1000,1);
title('Training Data');
[test_data,test_labels] =  generateMultiringDataset(2,10000,2);
title('Validation Data');

%% Initial Training
% Train a Gaussian kernel SVM with cross-validation to select
% hyperparameters that minimize probability of error

% Hyperparameter initialization
CList = 10.^linspace(-3,7,11);
sigmaList = 10.^linspace(-2,3,13);
lossVal = zeros(length(sigmaList),length(CList));

% Loop through all hyperparameters to find best combination
for sigmaCounter = 1:length(sigmaList)
    sigma = sigmaList(sigmaCounter);
    for CCounter = 1:length(CList)
        C = CList(CCounter);
        % Train SVM model for hyperparameter combination
        SVMModel = fitcsvm(train_data',train_labels,'BoxConstraint',C,...
            'KernelFunction','rbf','KernelScale',sigma);
        % Get cross-validated model (10-fold cross validation)
        CVSVMModel = crossval(SVMModel);
        % Store loss for SVM model
        lossVal(sigmaCounter,CCounter) = kfoldLoss(CVSVMModel);
    end
end

%% Final Training

% Find best sigma and C combination
minLoss = min(lossVal(:));
[sigmaBest_ind, CBest_ind] = find(lossVal == minLoss);

% Train final SVM and predict labels on test data
best_SVMModel = fitcsvm(train_data',train_labels,'BoxConstraint',...
    CList(CBest_ind),'KernelFunction','rbf','KernelScale',...
    sigmaList(sigmaBest_ind));
prediction = best_SVMModel.predict(test_data');

% Calculate accuracy of model
```

```
incorrect = sum(prediction' ~= test_labels);
accuracy = ((length(test_data) - incorrect)/length(test_data))*100

%% Plot Classified Labels

% Find and plot all correctly and incorrectly classified test samples
indINCORRECT = find(prediction' ~= test_labels); % incorrectly classified
indCORRECT = find(prediction' == test_labels);   % correctly classified
plot(test_data(1,indCORRECT),test_data(2,indCORRECT),'g.'); hold on;
plot(test_data(1,indINCORRECT),test_data(2,indINCORRECT),'r.'); axis equal;
title('Training Data (RED: Incorrectly Classified)');
xlabel('X_1'); ylabel('X_2');

%% Functions

function [data,labels] = generateMultiringDataset(numberOfClasses,
    numberOfSamples,fig)
    % Generates N samples from C ring-shaped class-conditional pdfs with equal
         priors

    % Randomly determine class labels for each sample
    thr = linspace(0,1,numberOfClasses+1); % split [0,1] into C equal length
        intervals
    u = rand(1,numberOfSamples); % generate N samples uniformly random in
        [0,1]
    labels = zeros(1,numberOfSamples);
    for l = 1:numberOfClasses
        ind_l = find(thr(l)<u & u<=thr(l+1));
        labels(ind_l) = repmat(l,1,length(ind_l));
    end

    a = [1:numberOfClasses].^3; b = repmat(2,1,numberOfClasses); % parameters
        of the Gamma pdf needed later
    % Generate data from appropriate rings
    % radius is drawn from Gamma(a,b), angle is uniform in [0,2pi]
    angle = 2*pi*rand(1,numberOfSamples);
    radius = zeros(1,numberOfSamples); % reserve space
    for l = 1:numberOfClasses
        ind_l = find(labels==l);
        radius(ind_l) = gamrnd(a(l),b(l),1,length(ind_l));
    end

    data = [radius.*cos(angle);radius.*sin(angle)];

    if 1
        colors = rand(numberOfClasses,3);
        figure(fig); clf;
        for l = 1:numberOfClasses
            ind_l = find(labels==l);
            plot(data(1,ind_l),data(2,ind_l),'.','MarkerFaceColor',colors(l,:)
                );
            axis equal; hold on;
            xlabel('X_1'); ylabel('X_2');
        end
    end
end
```

Appendix C: Question 3 Code

```
%% Take Home Exam 4: Question 3
% Anja Deric | April 13, 2020

% Clear all variables and load images in
clear all; close all;
filenames{1,1} = '3096_color.jpg';
filenames{1,2} = '42049_color.jpg';

for imageCounter = 1:2 %size(filenames,2)
    % Load and display original image
    imdata = imread(filenames{1,imageCounter});
    figure(1); subplot(size(filenames,2),3,(imageCounter-1)*3+1);
    imshow(imdata); title('Original Image');

    % Create and normalize feature vector
    [R,C,D] = size(imdata); N = R*C; imdata = double(imdata);
    rowIndices = [1:R]'*ones(1,C); colIndices = ones(R,1)*[1:C];
    % Initialize with row and column indices
    features = [rowIndices(:)';colIndices(:)'];
    % Add RGB values to feature vector
    for d = 1:D
        imdatad = imdata(:,:,d);
        features = [features;imdatad(:)'];
    end
    % Map all features to [0,1] range
    minf = min(features,[],2); maxf = max(features,[],2);
    ranges = maxf-minf;
    normalized = diag(ranges.^(-1))*(features-repmat(minf,1,N));

    % Fit 2-component GMM to image
    params = statset('MaxIter',1000);
    GMModel_2 = fitgmdist(normalized',2,'regularizationValue',1e-10, ...
                'Options',params);

    % Reshape and plot 2-component image
    labels = cluster(GMModel_2,normalized')==2;
    labelImage = reshape(labels,R,C);
    figure(1); subplot(size(filenames,2),3,(imageCounter-1)*3+2);
    imshow(uint8(labelImage*255)); title('2 Component Image');

    % 10-fold cross validation for 1-6 GMM component models
    kfold_split = cvpartition(length(normalized),'KFold',10);
    M = 6; K = 10; log_likelihood = zeros(M,K);
    for m = 1:M         % component model
        for k = 1:K     % cross-val

            % Get train and test data for each set
            train_index = kfold_split.training(k);
            test_index = kfold_split.test(k);
            train_data = normalized(:,find(train_index));
            test_data = normalized(:,find(test_index));

            % Fit GMModel to training data
            GMModel = fitgmdist(train_data',m,'regularizationValue',...
                1e-10,'Options',params);
            all_GMModels{m,k} = GMModel;

            % Calculate and store validation log-likelihood
            GMM_pdf = pdf(GMModel,test_data');
            log_likelihood(m,k) = sum(log(GMM_pdf));

        end
    end

    % Average all likelihoods and find best model order
    averagemleTest = mean(log_likelihood',1)
    [~, best_model] = max(averagemleTest);

    % Fit ideal GMModel to image and create labels
    best_GMModel = fitgmdist(normalized',best_model,'regularizationValue',...
        1e-10,'Options',params);
    best_labels = cluster(best_GMModel,normalized')-1;

    % Reshape image into original shape and plot
    best_labelImage = reshape(best_labels,R,C);
    figure(1); subplot(size(filenames,2),3,(imageCounter-1)*3+3);
    imshow(uint8(best_labelImage*255/(best_model-1)));
    title('Best Component Fit');

end
```