Assignment 4
Juan Pablo Bernal
06/11/2020

1. Question #1

Two datasets, where the data contained (x,y) pairs, were generated from the given Data Generation script. One set for training which contained 1000 samples and one for testing that contained 10000 samples (Figure 1).

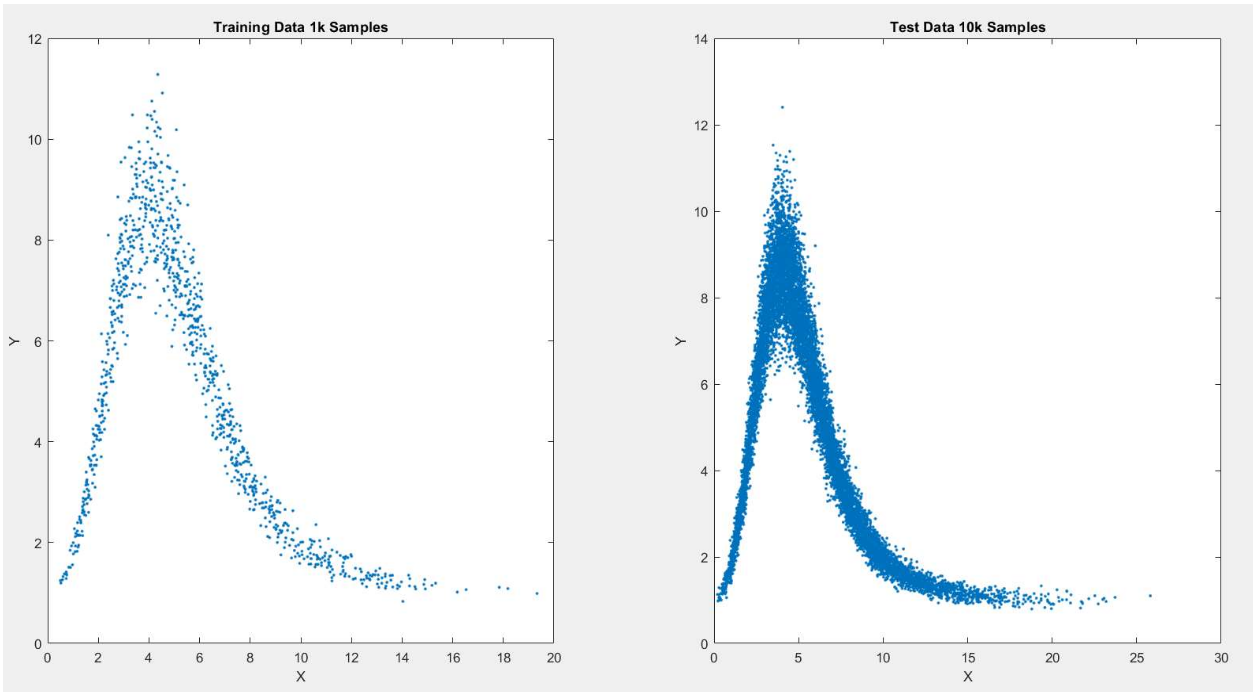

Figure 1. Training and Testing Datasets

The model believed to have generated these pairs was $y = f(x) + v$ where v is Gaussian noise with zero-mean and unit variance. With this model in mind, a neural network with 1 hidden layer and softplus ($ln(1+e^x)$) as the non-linear activation function was used to approximate $f(x)$. 10-fold Cross-validation was used to determine the optimal number of perceptrons in the 1st layer that would give the best approximation. Starting with one perceptron, this cross-validation was done by segmenting the training dataset in 10 subsets and each time using one of them as validation set and the rest as training data. With the training subset, a NN was trained (w and b parameter estimated with MLE) using fminsearch and MSE as target function. Once it was trained, the validation set was given as input to the NN, and the MSE was calculated with the equation $\sum_{i=1}^{N}(y_i - y^*_i)^2$ where y_i is the real value in the dataset and y^*_i is the NN output. After the 10-fold was finished the average MSE over the 10 trials was calculated. Next, the number of perceptrons was increased and everything was done once again. When the average MSE for the new number of perceptrons was calculated, it was compared with the previous average MSE. If it was smaller, the number of perceptrons was increased and the process ran once more. If it was greater than the previous one then the function would return the previous number of perceptrons (the one that gave the smallest MSE). After the optimal number of perceptrons was computed, the NN was trained with the entire training set, then it was applied to the 10k test dataset and the MSE was calculated using the same equation as before.

In one of the experiments that were run, the optimal number of perceptrons was found to be 6 and the minimum average MSE calculated during the cross-validation was 0.6461. After training and testing it the MSE on the test dataset was 0.7397, a little bit higher than the one found during cross-validation but this might be due to the difference in the number of samples between testing and training. In the end, both values are very close and the model can be considered a good approximation without overfitting it to the training data. In figure 2, the approximation given by the NN (in red) can be seen overlaid on the test data (in blue). The approximation follows the shape of the truth very closely. It just has some issues close to 0, this is because the soft-ramp function softplus might not be the best choice to approximate this kind of data.

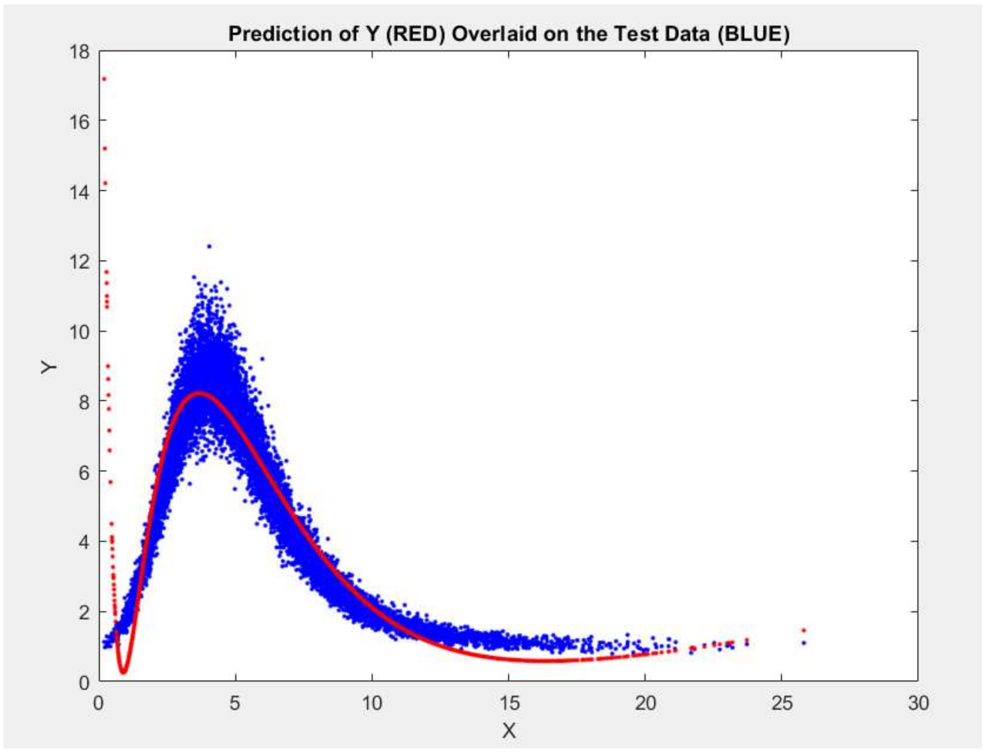

Figure 2. Model Approximation (RED) Overlaid on the Test Data (BLUE)

2. Question #2

Again the same two datasets were generated from the given generateMultiringDataset script, with the same number of samples for training and testing as in question 1. Figure 3 Shows the data distribution from the test dataset with one class in blue and the other one in orange. A Support vector machine was used to classify each sample to its respective class. The SVM uses a Gaussian/RBF kernel to make non-linear boundaries for each class. To estimate the sigma for the kernel and C for the box constraint (hyperparameters), 10-fold cross-validation was used once again. First, a set of 14 possible sigmas in the range of $[10^{-2}, 10^{3}]$, and a set 10 of possible values for c in the range $[10^{-1}, 10^{6}]$ were created. Then, for each combination of sigma and c, the training and validation sets were generated in the same way as in question 1. Next, the SVM was trained 10 times using Matlab's function fitcsvm specifying the Gaussian kernel and the box constraint. Then, the function SVM.predict was used in the validation set, and the number of incorrect classifications was recorded. With this information, the probability of incorrect classification for the SVM was calculated by dividing the number of incorrect classifications over the number of validation samples. After the 10 trials were completed, the average probability of error was calculated for the given combination of sigma and c. This value was recorded in a matrix whose rows represented the index of the sigma (in the possible sigmas set) and the columns represented the index of the value of c (in the possible c values set) that produced each particular probability of error. Finally, when all combinations were tested, the minimum average probability of error in the matrix was found and the sigma and c value for this minimum were chosen to be the optimal hyperparameters. Once they were selected, a final SVM was trained using the entire training dataset. Using the SVM.predict function once more, the test data was classified and the correct and incorrect classifications were recorded to calculate the probabilities of error and success of the model.

In one of the experiments ran, the optimal hyperparameters chosen were $sigma = 11.938$ and $C = 774.264$. With these values, the probability of error found in the test dataset was 0.0717 and the probability of success was 0.982. Figure 4 shows the correct classifications (Green dots) vs incorrect classifications (Red dots). All the incorrect classifications were located where the two classes overlap, an expected result since the boundaries are static and some unlucky samples from one class might fall in the wrong side of the boundary.

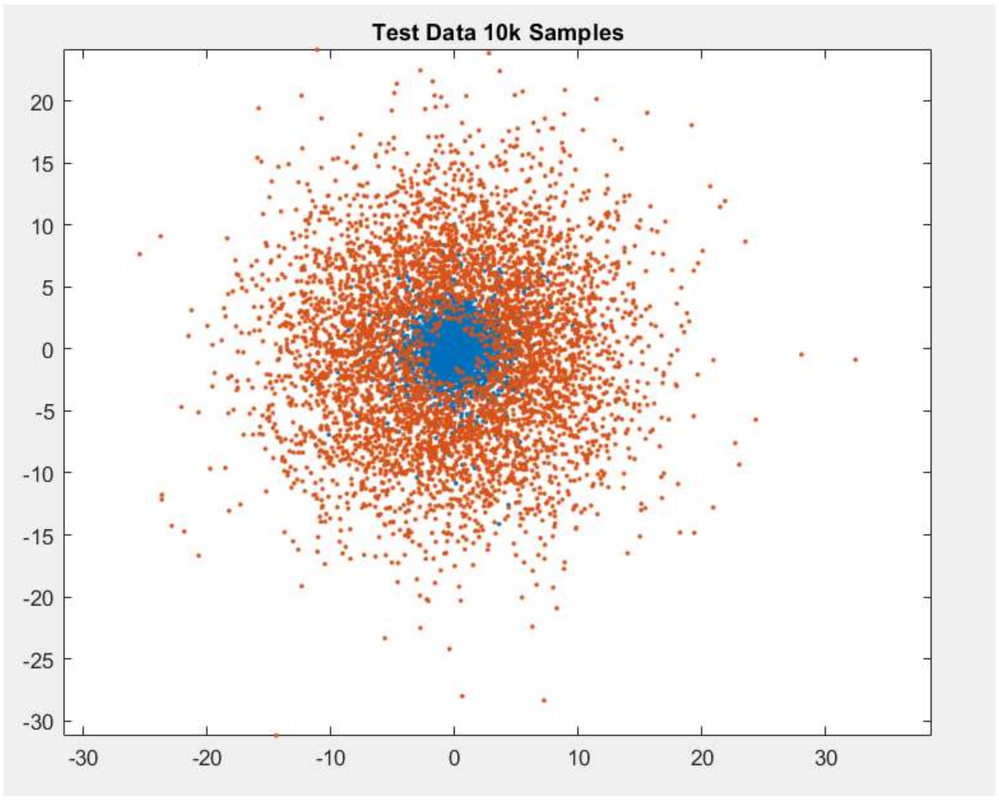

Figure 3. Data Distribution

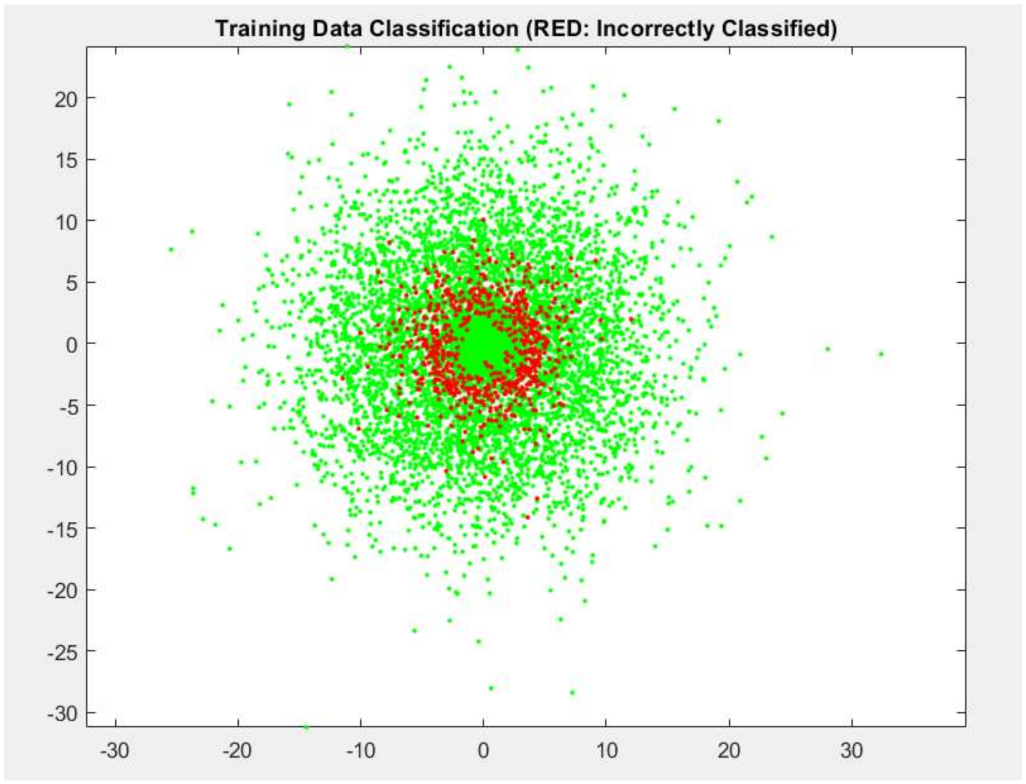

Figure 4. Classification Results

3. Question #3

The goal of this question was to segment two images using GMM-based clustering. The first image was an airplane in a cloudy sky background and a bird sitting in a branch with a clear sky background. First, the images were converted into a feature matrix where each row represented a pixel (raw feature vector). For each pixel, its row index, column index, red value, green value, and blue value (RGB) were recorded in a vector and then normalized to the range [0,1]. Using this matrix as data, a segmentation was done with a GMM-based model with 2 components. The GMM parameters were estimated using the ML algorithm from the Bilmes tutorial. The GMM-based segmentation uses the pdfs of the components in the Gaussian mixture as the class posterior for the clusters. Thus, a MAP classifier was used to segment the images $D = argmax_l P(L = l \mid x)$. For each pixel's normalized feature vector the likelihood under each component was calculated and it would be classified in the cluster that produced the maximum value. Once all the pixels in the image were classified, a synthetic image was generated by randomly assigning a color to each cluster, and each pixel modified to take its respective cluster color.
Images 1 and 2 show the result of the segmentation next to the original image. In the airplane image(Image 1.), the plane is one cluster (bright green) and the sky is another one (pale green). This can be considered a good clustering since it is differentiating the main object from the background. Moreover, in the bird image (Image 2.), the clusters are the bird with the branch and the darkened corners (orange), and the other one the sky (green). This clustering has some issues due to the darkened corners that should be considered as sky instead as part of the foreground (bird and branch). It might be that the model grouped dark things together therefore these dark corners are more similar to the bird than to the sky.
To fix these issues and see if both models could be improved, 10-fold cross-validation was used to find the best number of clusters to segment each image. This cross-validation was done in the same way as in question 1. This time we wanted to estimate the number of Gaussian components instead of the number of perceptrons. With 1 component, using the training subset the GMM parameters were estimated using the same ML algorithm. Once they were estimated, the log-likelihood of the validation set under the estimated GMM was computed. This was done 10 times and the average log-likelihood was computed. Then the number of components was increased and everything was done again. This was done until the average log-likelihood for a new number of GMM components was less than the previous one. And the function returned the number of components that produce the maximum average log-likelihood. With the number of components for each image the segmentation process was done again and the results are shown in images 3 and 4. The airplane looks almost the same, it was divided in 3 segments, the airplane (pink), left half of the sky which has less white clouds (dark green) and the right half of the sky which contained more clouds (blue). In the end both segmentations were good and not much improvement was made by doing the

cross validation. On the other hand, the bird image looks much better, this time it was segmented in 4clusters, one for the bird and branch (pale green), another one for the sky (yellow), another one for the darkened corners in the left (purple) and one for the darkened corners in the right (brown). The only thing is that since this picture is very pixelated in the edges of the objects, the program considered that they were part of the brown cluster. Moreover, multiple experiments were run to see if the mistakes the model made were due to a local minimum in the parameter estimation but every experiment would come out similar. The airplane was always bad and the bird always had the misclassification of the pixelated regions.

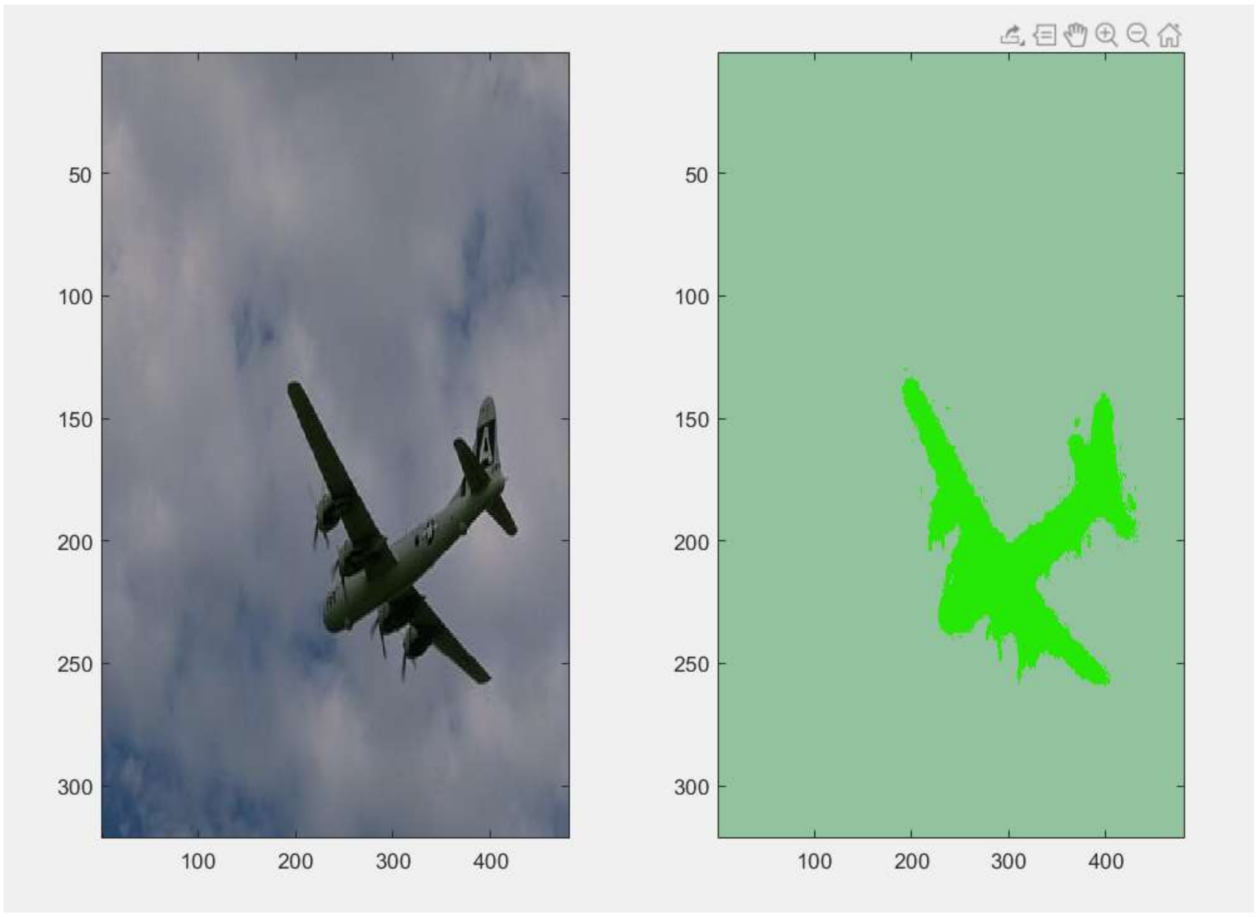

Image 1. Airplane Image Segmentation Using 2-Component GMM

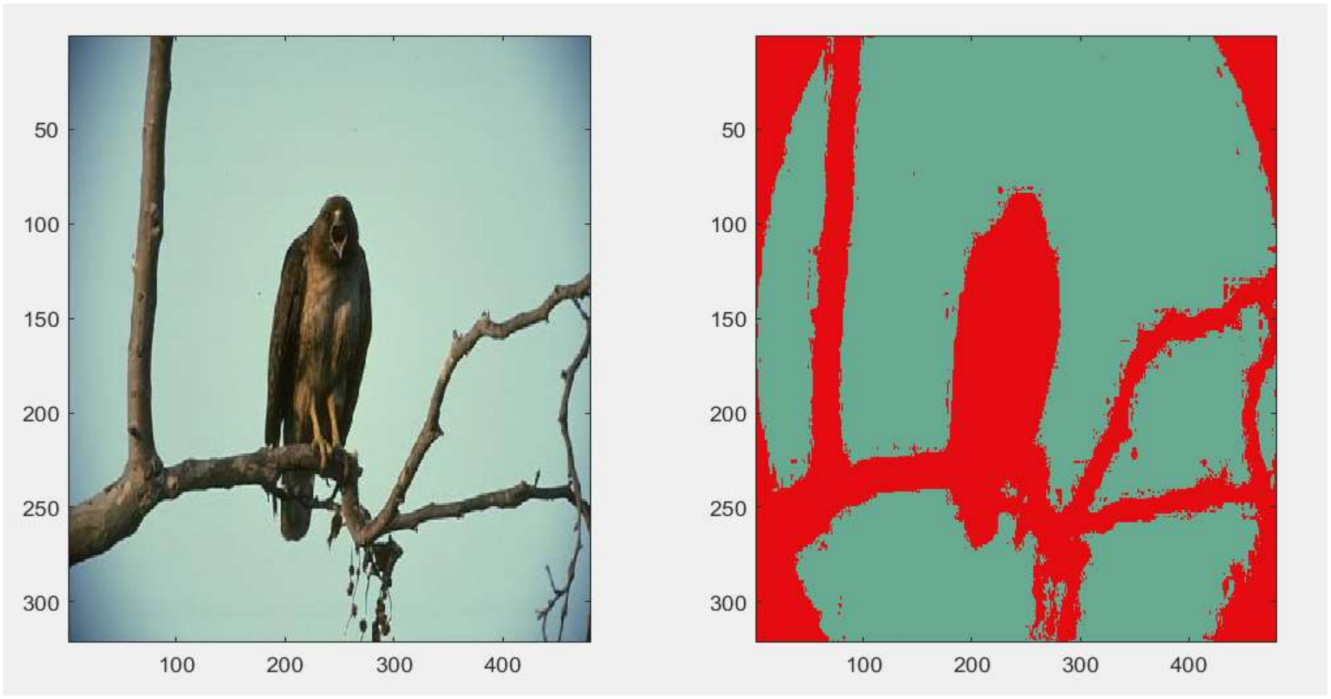

Image 2. Bird Image Segmentation Using 2-Component GMM

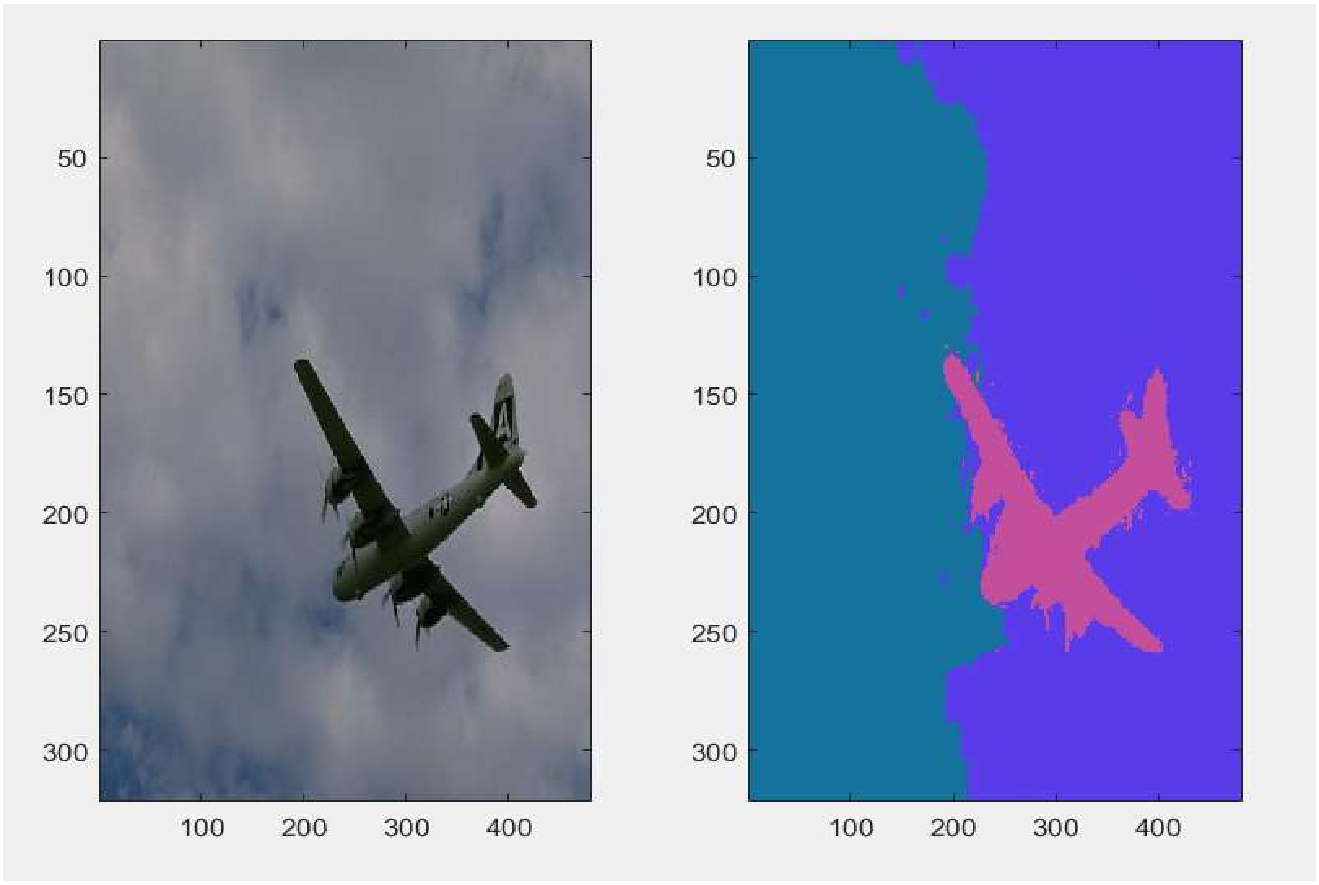

Image 3. Airplane Image Segmentation Using 3-Component GMM

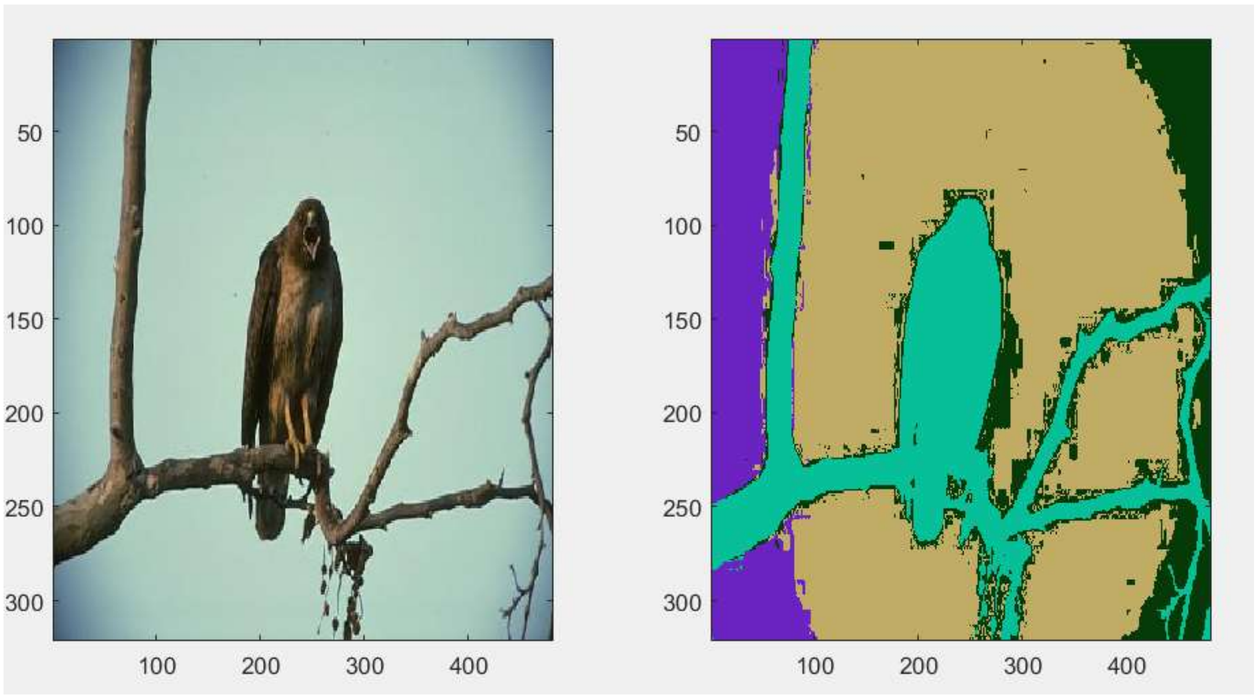

Image 4. Bird Image Segmentation Using 4-Component GMM

Citations

The GMM ML algorithm (called MLEGMM in the code) and evalGMM function (called evalgmm in this code) was extracted from MLforGMM.m code found in the G/drive.

The SVM in question 2 format and some general equations were extracted from the svmKfoldCrossValidation.m script given in the G/drive

For the neural network in question 1, the format of the code and the overall solution were based on the given code mleMLPwAWGN.m but nothing was directly copied from it.

Appendix A
Code Question #1

```
clear all, close all
%Datasets
[xtrain,ytrain] = exam4q1_generateData(1000,1);
[xtest,ytest] = exam4q1_generateData(10000,2);

%number of perceptrons
Np = C_V(xtrain,ytrain);

%train and test the NN
p = NNtrain(xtrain,ytrain,Np,10000);
[TE,H] = NNtest(p,xtest,ytest,10000);

figure(3), plot(xtest,ytest,'b.'),hold on,
xlabel('X'), ylabel('Y'), title('Prediction of y (RED) over Test
Data(BLUE)');
figure(3), plot(xtest,H,'r.'),hold off;

figure (4),subplot(1,2,1),plot(xtrain,ytrain,'.'),title ('Training Data 1k
Samples'),
subplot(1,2,2),plot(xtest,ytest,'.'),title('Test Data 10k Samples');

%Cross-Validation with 10-fold to estimate number of perceptrons
function order = C_V(x,y)
    s = length(x)/10;
    done = 0;
    Np = 0;
    currentScore = inf;

    while ~done
        Np = Np+1;
        orderMSE = 0;
        %10-fold
        for k = 1:10
            x_validate = [];
            x_train = [];
            y_validate = [];
            y_train = [];
            if k == 1
               x_validate = x([1:(k*s)]);
               x_train = x([(k*s+1):end]);
               y_validate = y([1:(k*s)]);
               y_train = y([(k*s+1):end]);

            elseif k == 10
               x_validate = x([(length(x)-s+1):end]);
               x_train = x([1:(length(x)-s)]);
               y_validate = y([(length(x)-s+1):end]);
               y_train = y([1:(length(x)-s)]);

            else
                x_train = x([1:(((k-1)*s))]);
                x_validate = x([(((k-1)*s)+1):(k*s)]);
                x_train = [x_train x([(k*s+1):end])];
                y_train = y([1:(((k-1)*s))]);
                y_validate = y([(((k-1)*s)+1):(k*s)]);
                y_train = [y_train y([(k*s+1):end])];
            end
            p = NNtrain(x_train,y_train,Np,length(x_train));
            orderMSE = orderMSE +
NNtest(p,x_validate,y_validate,length(x_validate));
            k
        end
        NewScore  = orderMSE/10
        if NewScore > currentScore;
            done = 1;
        else
            currentScore = NewScore;
        end
        Np
    end
    order = Np-1;
    currentScore
end
%Test the NN estimated with a new 10k samples dataset
function [E,h] = NNtest(p,x,y,N)
    p.w2 = p.w2';
    h = MLP1L(x,p);
    E = sum((y-h).*(y-h),2)/N;
end
```

```matlab
%Train the NN by estimating its parameters
function p = NNtrain(x,y,Np,N)

    for t = 1:10
        %initialize the parameters randomly
        p.w1 = rand(Np,1);
        p.b1 = rand(Np,1);
        p.w2 = rand(1,Np);
        p.b2 = rand(1,1);
        vecPinit = [p.w1(:);p.b1(:);p.w2(:);p.b2(:)];
        options = optimset('MaxIter',1e15*length(vecPinit));
        [Ep, val] = fminsearch (@(Ep)(MSE(x,Ep,y,Np,N)),vecPinit,options);

        p.w1 = Ep(1:Np);
        p.b1 = Ep((Np+1):(2*Np));
        p.w2 = Ep((2*Np+1):(3*Np));
        p.b2 = Ep((3*Np+1):end);
        scores(t) = val;
        allp(t) = p;
    end
    [~,Min] = min(scores);

    p = allp(Min);
end
%MSE calculator and function for fminsearch
function E = MSE(D,vecp,y,Np,N)
    p.w1 = vecp(1:Np);
    p.b1 = vecp((Np+1):(2*Np));
    p.w2 = (vecp((2*Np+1):(3*Np)))';
    p.b2 = vecp((3*Np+1):end);
    h = MLP1L(D,p);
    %MSE
    E = sum((y-h).*(y-h))/N;
end
%1 layer MLP
function h = MLP1L(D,p)
    %First Layer softplus activation function
    softP = @(r) (1+exp(r));
    logic = @(r) 1./(1+exp(-r));
    x = p.w1*D + p.b1;
    %z = logic(x);
    z = softP(x);
    h = p.w2*z + p.b2;
end
%Generate Data
function [x,y] = exam4q1_generateData(N,i)
x = gamrnd(3,2,1,N);
z = exp((x.^2).*exp(-x/2));
v = lognrnd(0,0.1,1,N);
y = v.*z;
figure(i), plot(x,y,'b.'),hold on,
xlabel('x'); ylabel('y');
end
```

Appendix B
Code Question #2

```
clear all, close all;
%Datasets
[xtrain,labeltrain] = generateMultiringDataset(2,1000,1);
[xtest,labeltest] = generateMultiringDataset(2,10000,2);

%Estimate hyperparameters
[sigma,c] = C_V(xtrain,labeltrain);

%Train and validate the model
svm =
fitcsvm(xtrain',labeltrain,'BoxConstraint',c,'KernelFunction','rbf','kernel
Scale',sigma);
validation = svm.predict(xtest')';
%Find the probability of error and success
indCORRECT = find(labeltest==validation);
indInCORRECT = find(labeltest~=validation);
Pcorrect = length(indCORRECT)/length(labeltest);
PIncorrect = length(indInCORRECT)/length(labeltest);

figure(3),
plot(xtest(1,indCORRECT),xtest(2,indCORRECT),'g.'), hold on,
plot(xtest(1,indInCORRECT),xtest(2,indInCORRECT),'r.'), axis equal,
title('Training Data (RED: Incorrectly Classified)'),

 %Cross-Validation with 10-fold to estimate hyperparameters of the SVM
```

```
function [Esigma,Ec] = C_V(x,y)
    s = length(x)/10;
    sigmas = 10.^linspace(-2,3,14); Cs = 10.^linspace(-1,6,10);
    for sig = 1:length(sigmas)
        sigma = sigmas(sig);
        for i = 1:length(Cs)
            c = Cs(i);
            %10-fold
            for k = 1:10
                x_validate = [];
                x_train = [];
                y_validate = [];
                y_train = [];
                if k == 1
                   x_validate = x(:,[1:(k*s)]);
                   x_train = x(:,[(k*s+1):end]);
                   y_validate = y([1:(k*s)]);
                   y_train = y([(k*s+1):end]);

                elseif k == 10
                   x_validate = x(:,[(length(x)-s+1):end]);
                   x_train = x(:,[1:(length(x)-s)]);
                   y_validate = y([(length(x)-s+1):end]);
                   y_train = y([1:(length(x)-s)]);

                else
                    x_train = x(:,[1:(((k-1)*s))]);
                    x_validate = x(:,[(((k-1)*s)+1):(k*s)]);
                    x_train = [x_train x(:,[(k*s+1):end])];
                    y_train = y([1:(((k-1)*s))]);
                    y_validate = y([(((k-1)*s)+1):(k*s)]);
                    y_train = [y_train y([(k*s+1):end])];
                end

                svm =
fitcsvm(x_train',y_train,'BoxConstraint',c,'KernelFunction','rbf','kernelSc
ale',sigma);
                validation = svm.predict(x_validate')';
                indInCORRECT = find(y_validate ~= validation);
                PIncorrect(k)=length(indInCORRECT)/length(y_validate);
            end
            scores(sig,i)  = sum(PIncorrect)/10;
            sig
            i
        end
    end
    MinP = min(scores,[],'all');
    [row,col] = find(scores == MinP);
    Esigma = sigmas(row(1));
    Ec = Cs(col(1));
end
%given function to generate data
function [data,labels] =
generateMultiringDataset(numberOfClasses,numberOfSamples,i)

C = numberOfClasses;
N = numberOfSamples;
% Generates N samples from C ring-shaped
% class-conditional pdfs with equal priors

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

a = [1:C].^2.5; b = repmat(1.7,1,C); % 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,N);
radius = zeros(1,N); % reserve space
for l = 1:C
    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(C,3);
    figure(i), clf,
    for l = 1:C
        ind_l = find(labels==l);

plot(data(1,ind_l),data(2,ind_l),'.','MarkerFaceColor',colors(l,:)); axis
equal, hold on,
    end
end
end
```

Appendix C
Code Question #3

```
clear all, close all;

airplane = imread('3096_color.jpg');
bird = imread('42049_color.jpg');

airplanePixelVec = genPixelVec(airplane);
birdPixelVec = genPixelVec(bird);

airplaneNormPixelVec = airplanePixelVec/max(airplanePixelVec,[],'all');
birdNormPixelVec = birdPixelVec/max(birdPixelVec,[],'all');

%load('imageVectors');

%Using 2 Gaussian components to segment the images
[Aalpha,Amu,Asigma] = MLEGMM(airplaneNormPixelVec',1e-2,2);
Asegment = MAP(airplaneNormPixelVec',Aalpha,Amu,Asigma,2);
AirplaneSeIm1 = genSegImg(airplanePixelVec,Asegment,2);
figure(1),subplot(1,2,1),
image(airplane),subplot(1,2,2),image(AirplaneSeIm1)

[Balpha,Bmu,Bsigma] = MLEGMM(birdNormPixelVec',5e-2,2);
Bsegment = MAP(birdNormPixelVec',Balpha,Bmu,Bsigma,2);
BirdSeIm1 = genSegImg(birdPixelVec,Bsegment,2);
figure(2),subplot(1,2,1), image(bird),subplot(1,2,2),image(BirdSeIm1)

%Using Cross-Validation to find optimal number of components, train the
%model using test data and segment the image
```

```
ANgmm = C_V(airplaneNormPixelVec');
[Aalpha,Amu,Asigma] = MLEGMM(airplaneNormPixelVec',5e-2,ANgmm);
Asegment = MAP(airplaneNormPixelVec',Aalpha,Amu,Asigma,ANgmm);
AirplaneSeIm2 = genSegImg(airplanePixelVec,Asegment,ANgmm);
figure(3),subplot(1,2,1),
image(airplane),subplot(1,2,2),image(AirplaneSeIm2)


BNgmm = C_V(birdNormPixelVec');
[Balpha,Bmu,Bsigma] = MLEGMM(birdNormPixelVec',5e-2,BNgmm);
Bsegment = MAP(birdNormPixelVec',Balpha,Bmu,Bsigma,BNgmm);
BirdSeIm2 = genSegImg(birdPixelVec,Bsegment,BNgmm);
figure(4),subplot(1,2,1), image(bird),subplot(1,2,2),image(BirdSeIm2)

%Cross-Validation to find optimal number of gaussian components
function Ngmm = C_V(x)
    s = round(length(x)/10);
    done = 0;
    t = 0;
    currentScore = -inf;
    while ~done
        t = t+1;
        score = 0;
        %10-fold
        for k = 1:10
            x_validate = [];
            x_train = [];
            y_validate = [];
            y_train = [];
            if k == 1
                x_validate = x(:,[1:(k*s)]);
                x_train = x(:,[(k*s+1):end]);
            elseif k == 10
                x_validate = x(:,[(length(x)-s+1):end]);
                x_train = x(:,[1:(length(x)-s)]);
            else
                x_train = x(:,[1:(((k-1)*s))]);
                x_validate = x(:,[(((k-1)*s)+1):(k*s)]);
                x_train = [x_train x(:,[(k*s+1):end])];
            end

            [Ealpha, Emu, Esigma] = MLEGMM(x_train,1e-1,t);
            score = score +
sum(log(sum(evalgmm(x_validate,Emu,Esigma,Ealpha,t))));
        end
        NewScore = sum(score)/10;

        if NewScore < currentScore
            done = 1;
            Ngmm = t-1;
        else
            currentScore = NewScore;
        end
    end
end
%generate an image where each color represents the classification of a
%pixel
function Simage = genSegImg(pixelVec,segment,M)
    colors = round(rand(M,3)*255);
    for i = 1:length(pixelVec)
        for c = 1:M
            if segment(i) == c
                Simage(pixelVec(i,1), pixelVec(i,2),1) = colors(c,1);
                Simage(pixelVec(i,1), pixelVec(i,2),2) = colors(c,2);
                Simage(pixelVec(i,1), pixelVec(i,2),3) = colors(c,3);
                break;
            end
        end
    end
    Simage = uint8(round(Simage));
end
%MAP classification rule to assign each pixel to a cluster
function segment = MAP(x,alpha,mu,sigma,M)

    score = evalgmm(x,mu,sigma,alpha,M);

     [~,segment] = max(score);

end
%generate the raw feature vector of an image
function pixelVec = genPixelVec(image)

    s = size(image);
    pixelVec = [];
    for row = 1:s(1)
        for col = 1:s(2)
            pixel(1) = row;
            pixel(2) = col;
            pixel(3) = image(row,col,1);
            pixel(4) = image(row,col,2);
            pixel(5) = image(row,col,3);
            pixelVec = [pixelVec; pixel];
        end
    end
end
%ML algorithm to estimate GMM parameters
function [Ealpha, Emu, Esigma] = MLEGMM(x,delta,M)

    converged = 0; d = 5; regWeight = 1e-10; N = length(x);

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

    while ~converged & t <=300

        for l = 1:M
            temp(l,:) =
repmat(Ealpha(l),1,N).*(mvnpdf(x',Emu(:,l)',Esigma(:,:,l)))';
        end

        plgivenx = temp./sum(temp,1);
        alphaNew = mean(plgivenx,2);
        w = plgivenx./repmat(sum(plgivenx,2),1,N);
        muNew = x*w';
        for l = 1:M
            v = x-repmat(muNew(:,l),1,N);
            u = repmat(w(l,:),d,1).*v;
            SigmaNew(:,:,l) = u*v' + regWeight*eye(d,d); % adding a small
regularization term
        end
        Dalpha = sum(abs(alphaNew-Ealpha'));
        Dmu = sum(sum(abs(muNew-Emu)));
        DSigma = sum(sum(abs(abs(SigmaNew-Esigma))));
        converged = ((Dalpha+Dmu+DSigma)<delta); % Check if converged
        Ealpha = alphaNew; Emu = muNew; Esigma = SigmaNew;
        t = t+1;
    end
end
%Evaluate the pdf of each of the components in the GMM
 function pdf = evalgmm(x,mu,sigma,alpha,M)

    for i = 1:M % evaluate the GMM on the grid
        pdf(i,:) = (mvnpdf(x',mu(:,i)',sigma(:,:,i)))'*alpha(i);
    end
end
```