EECE 5644 Homework 4 Report

Shijin Wang

Q1.
Description:

In this question, we are asked to train and test a single hidden MLP to estimate the MSE and select the best number of perceptrons. Then, using the whole training data set to train and estimate the MSE on the test data set with the selected number of perceptrons.

Process:

The overall model function is : $h(x, \theta) = m(d + c\,\alpha(b + Ax))$

The parameters need to be optimized : A. b. c. d.

MSE Rule:

$$MSE(\bar{X}) = E((\bar{X} - \mu)^2) = \left(\frac{\sigma}{\sqrt{n}}\right)^2$$

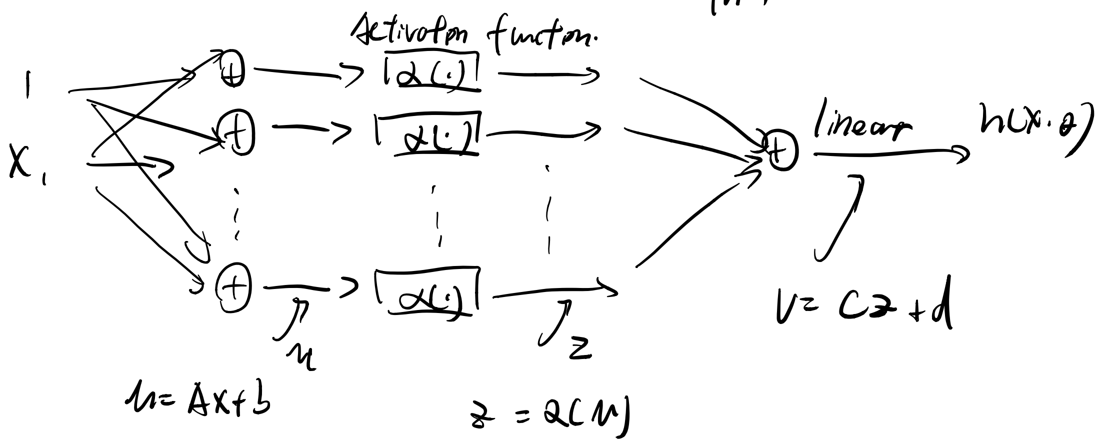

Figure 1. The structure of MLP

The first step is to split the dataset into two sets. Since the data is in two dimensional, the first row (x1) will be the training data and the second row will be the testing data(x2).

In order to perform the 10 fold cross-validation, the index of the training set and validation set was split into 10 blocks. For each perceptron from 1 to 10, 90% of the training dataset is used for training and the rest is used for validation. All the parameters for H function are randomly initialized. The fminsearch library is used to achieve the objective which is to minimize the square error for parameters by comparing the input labels with the output from the linear output layer. Meanwhile, the MSE is calculated by the MSE equation in figure 1. An array named MSEAverage is created to store the average MSE of each perceptron.

Lastly, after all the MSE were generated from 10-fold cross-validation, the perceptron with the lowest MSE is selected which then passes to the final MLP model trained by the entire training dataset. Finally, apply the testing dataset to this model and calculate the final MSE with the labels.

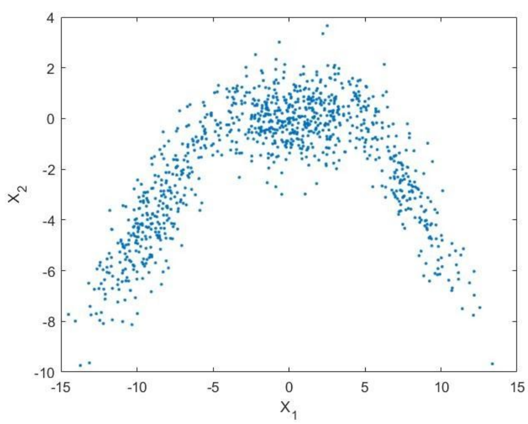

Figure 2. 1000 training samples

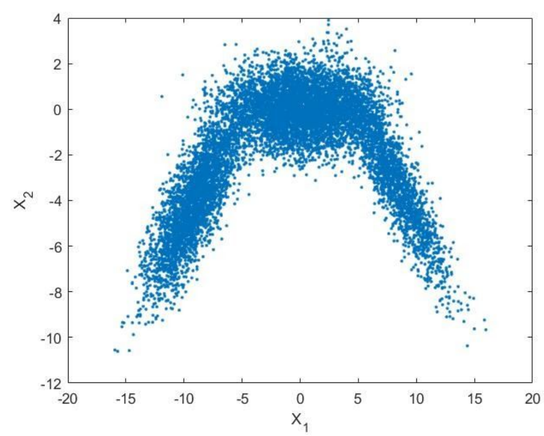

Figure 3. 10000 testing samples

MSEAverage =

4.6990 1.7449 1.2754 1.1536 1.1066 1.1450 1.1448 1.1064 1.1143 1.1114

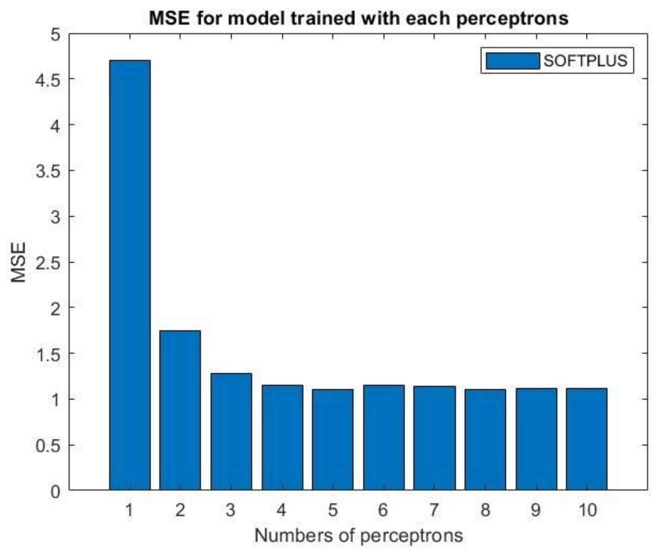

Figure 4. MSE for an MLP model trained from 1 to 10 perceptrons with the training dataset

MSEFinal = 1.2634

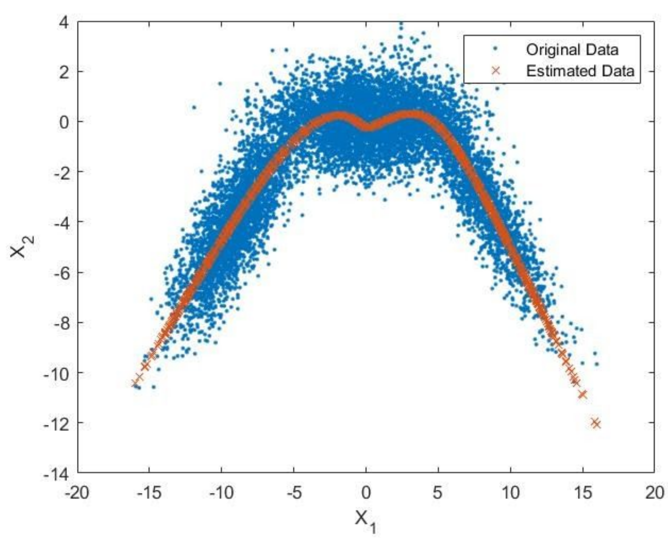

Figure 5. The estimated result on the testing dataset with 8 perceptrons

Conclusion:

In figure 4, the highest MSE happens when the perceptron equals to one. The MSE is around the same after three perceptrons. In this experiment, 8 perceptrons were selected because of the lowest MSE. In figure 5, we can see that the estimated result fits well with the testing dataset.

Q2.

Description:

Train and evaluate a support vector machine classifier with a Gaussian kernel on 1000 training dataset, find the best box constraint hyperparameter C and the Gaussian kernel width parameter Sigma. Then, train a final SVM using this combination of hyperparameters with the entire dataset and evaluate the performance on the testing dataset.

Process:

The basic idea of SVM is to find a function $f(x)$ that has at most ε deviation from the actually obtained targets y_i for all the training data, and at the same time is as flat as possible.

$f(x) = xw + b$, where $w = \sum_i \alpha_i y_i x_i$ (sum of training samples)

Since the dataset is not linear we need a kernel, specifically, Gaussian / RBF kernel. $K(x_1, x_2) = e^{-\|x_1 - x_2\|^2 / 2\sigma^2}$. So that $f(x_j) = \sum_i \alpha_i y_i e^{-\|x_i - x_j\|^2/2\sigma^2} + b$.

Apply this to the non-separable case: where we need to maximize the margin, minimize the errors: Set ε_i to be the constraints of the error. $\sum_i \varepsilon_i$ is the upper bound of the training error. We also assign C to $\|w\|^2/2 + C(\sum_i \varepsilon_i)^k$ which is need to control the penalty of the error. So that

$L_p = \frac{1}{2}\|w\|^2 + C\sum_i \varepsilon_i - \sum_i \alpha_i \{y_i(e^{-\|x_i - x_j\|^2/2\sigma^2} + b) - 1 + \varepsilon_i\} - \sum_i \mu_i \varepsilon_i$. We need to find the best C and σ

In order to find the best combination of C and Sigma, it is necessary to generate those values from a large but reasonable range. The list of C is created by selecting 11 numbers from 10^-1 to 10^9, and the list of Sigma is created by selecting 13 numbers from 10^-2 to 10^3. The 10 fold cross-validation procedure is exactly the same as Question 1. The “fitcsvm” library is used to train the support vector machine. After comparing the label set with the predicted value, the number of correctly classified data was stored in an array which can be used to calculate the percent of correctness and find the best combination of C and Sigma. Then, applying the best C and Sigma to the final SVM model which is trained by the entire dataset, predicting the label of the test dataset, the final percentage of correct classification can be calculated.

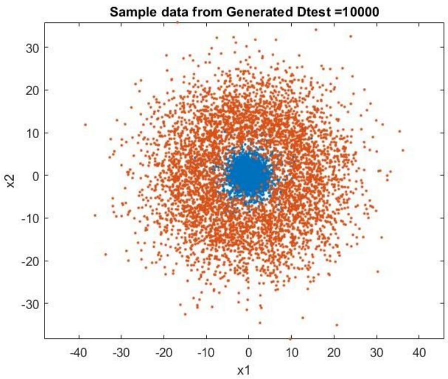

Figure 6. 1000 training samples

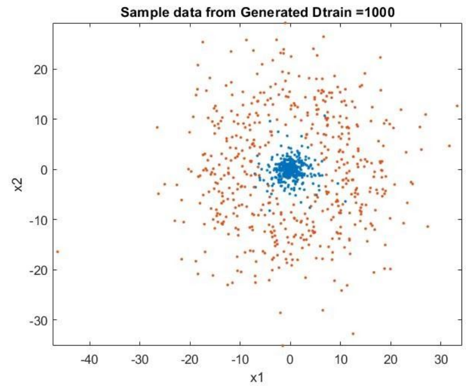

Figure 7. 10000 testing samples

Best Sigma = 1.0e+03 * 0.0012 (6th) Best C value = 1.0e+00

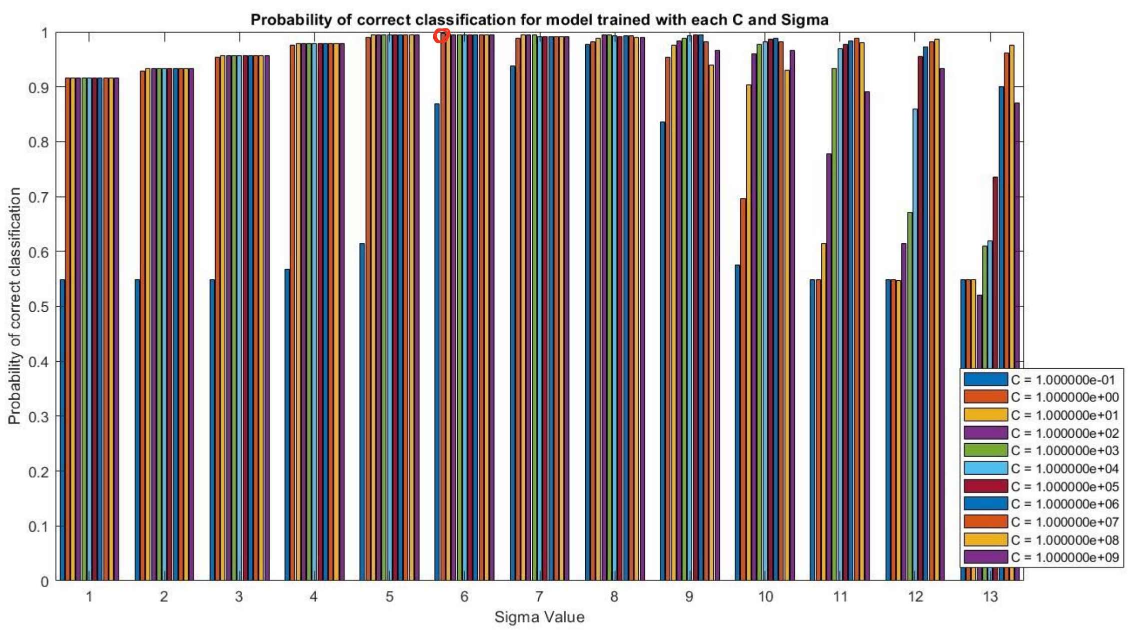

Figure 8. The average percentage of accuracy for a model trained with each C and sigma hyperparameter

Best Sigma = 1.0e+03 * 0.0012 Best C value = 1.0e+00 (2th)

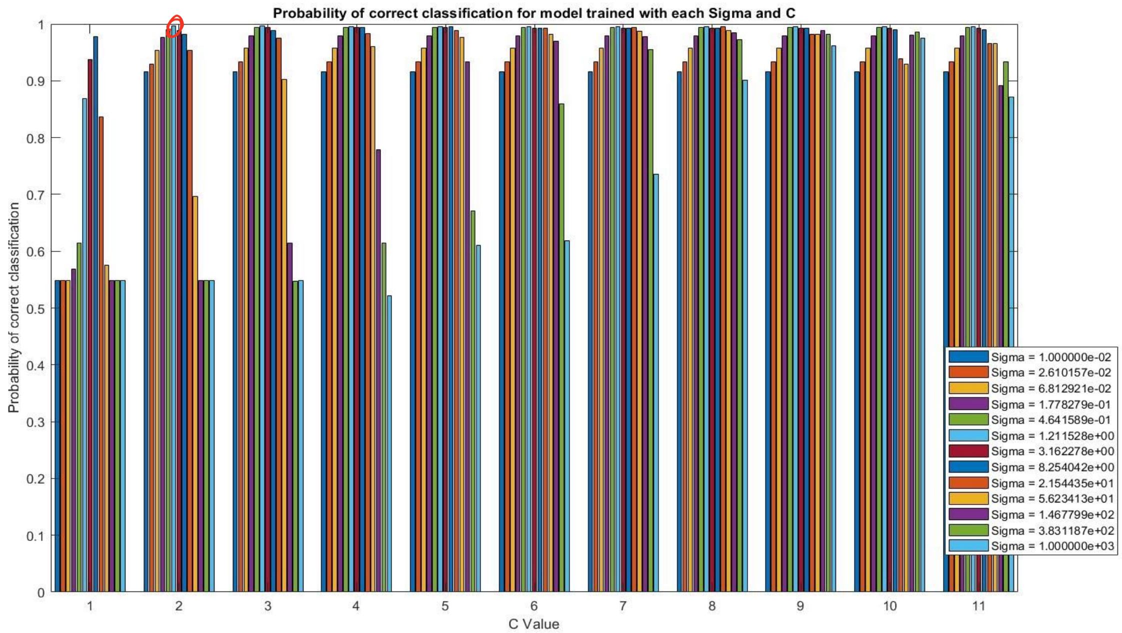

Figure 9. The average percentage of accuracy for a model trained with each sigma and C hyperparameter

PCorrect =

Columns 1 through 11

0.5480	0.5480	0.5490	0.5680	0.6150	0.8690	0.9380	0.9780	0.8360	0.5760	0.5480
0.9160	0.9290	0.9530	0.9760	0.9900	0.9970	0.9880	0.9820	0.9540	0.6960	0.5480
0.9160	0.9330	0.9570	0.9790	0.9940	0.9960	0.9940	0.9890	0.9750	0.9030	0.6150
0.9160	0.9330	0.9570	0.9790	0.9940	0.9950	0.9940	0.9940	0.9830	0.9600	0.7780
0.9160	0.9330	0.9570	0.9790	0.9940	0.9950	0.9940	0.9950	0.9880	0.9770	0.9340
0.9160	0.9330	0.9570	0.9790	0.9940	0.9950	0.9920	0.9930	0.9930	0.9820	0.9700
0.9160	0.9330	0.9570	0.9790	0.9940	0.9950	0.9920	0.9920	0.9940	0.9870	0.9780
0.9160	0.9330	0.9570	0.9790	0.9940	0.9950	0.9920	0.9930	0.9950	0.9880	0.9840
0.9160	0.9330	0.9570	0.9790	0.9940	0.9950	0.9920	0.9930	0.9820	0.9820	0.9890
0.9160	0.9330	0.9570	0.9790	0.9940	0.9950	0.9920	0.9900	0.9390	0.9300	0.9800
0.9160	0.9330	0.9570	0.9790	0.9940	0.9950	0.9920	0.9900	0.9660	0.9660	0.8910

Columns 12 through 13

0.5480	0.5480
0.5480	0.5480
0.5470	0.5480
0.6140	0.5210
0.6710	0.6100
0.8600	0.6190
0.9550	0.7360
0.9720	0.9010
0.9820	0.9610
0.9860	0.9750
0.9340	0.8710

Figure 10. The average percentage of accuracy matrix (The row is for C value, the column is for sigma value)

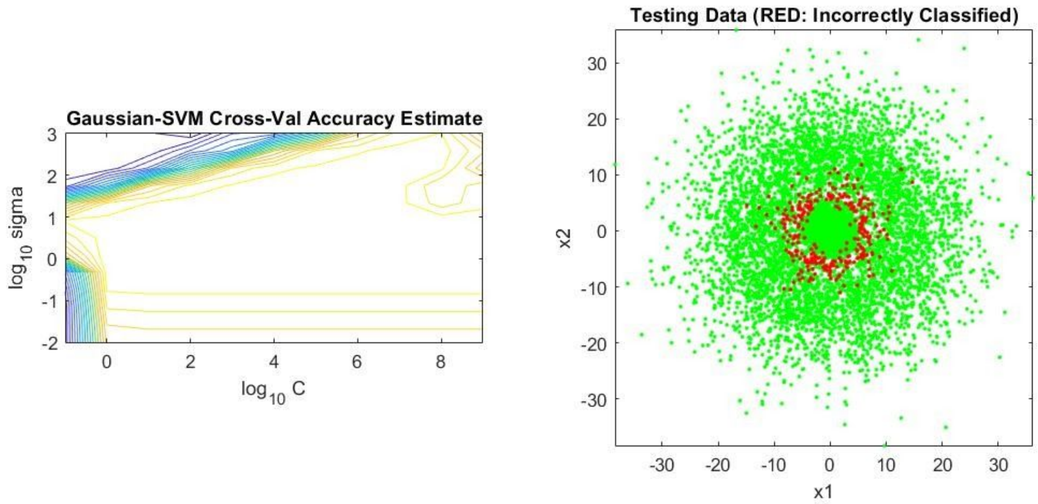

Figure 11. Gaussian-SVM Cross-Val Accuracy Estimate

pTrainingError = 0.0367

Conclusion:

From figure 8, we can see that the highest percentage of accuracy is around 7th Sigma value(1.0e+03 * 0.0012). The percentage of accuracy goes down as the sigma value is too small or too large which can be further proofed in figure 9. No matter which C value is, the accuracy does not drop too low if the sigma value is around 7th. This observation is the same as what the professor mentioned in the class: If sigma is too small, the kernels will be overfitting, If the sigma is too large, the kernels will be over smoothing. From the accuracy matrix in figure 10, we can see that the highest percentage of accuracy is achieved by 6th Sigma = 1.0e+03 * 0.0012, and 2th C value = 1.0e+00.

In Figure 11, the final SVM model trained with the entire training dataset has a relatively low percentage of training error which is 3.67%.

Q3.

Description:

Generate a 5-dimensional feature vector from each color image. Firstly, fit this GMM with 2-components using maximum parameter estimation (EM algorithm). Secondly, using 10-fold cross-validation find the best number of clusters. Lastly, fit a new GMM with this best number of components.

Process:

In this problem, the training images need to be preprocessed to the 5-dimensional feature vector by arranging the row index, column index, red value, green value, and blue value to each row. Then, perform a linear scaling by divided each feature by its range.

First, the GMM order is set to two. The fitgmdist library which uses the EM algorithm is used to perform the maximum likelihood parameter estimation. Generate the posterior probability from the GMM for each Gaussian component, then label the maximum posterior as the index of the new image. Reshape the image with the original row and column. Finally, use the imagesc to display the new image with colormap. Compute each GMM order (2-5) for each 10-fold cross-validation procedure which was implemented the same way as Question 1 and Question 2. Then use the fitgmdist library to find the best alpha, mu, and sigma for each model order. Use evalGMM function to evaluate with the validation dataset and then record the sum of this log-likelihood.

After 10 fold cross-validation, an average sum log-likelihood was calculated for each component. The best GMM order will be selected by the component with the highest log-likelihood. Then, passing this GMM order to the fitgmdist (EM algorithm) again and use the same steps for part 1 to display the segmented images.

Figure 12. Original plane color image

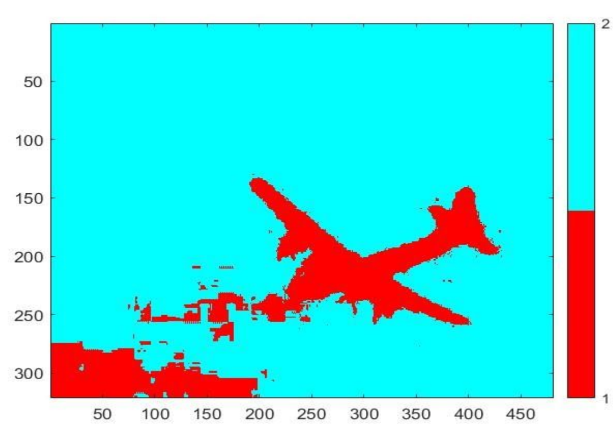

Figure 14. Plane color image segmented into two parts

Figure 13. Original bird color image

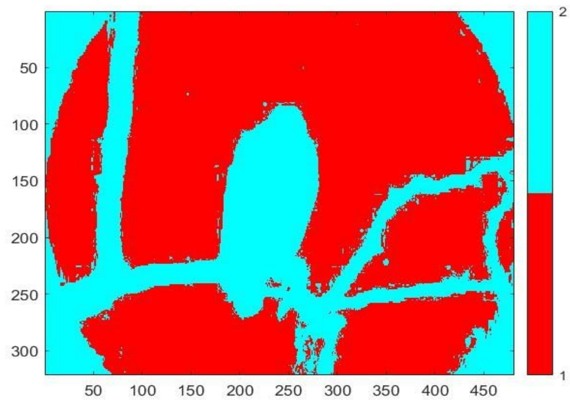

Figure 15. Bird color image segmented into two parts

Best GMM clusters = 4

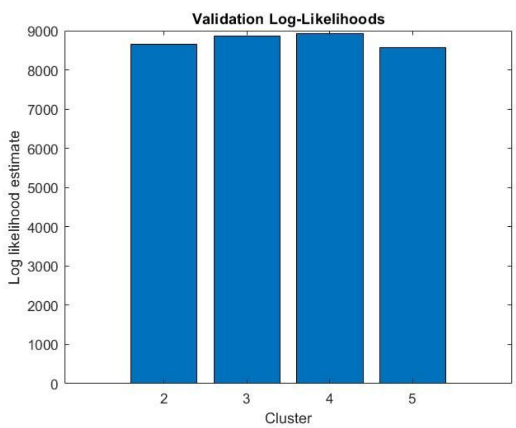

Figure 16. Percentage accuracy of a model trained with a Plane image from 2 to 5 clusters

Best GMM clusters = 3

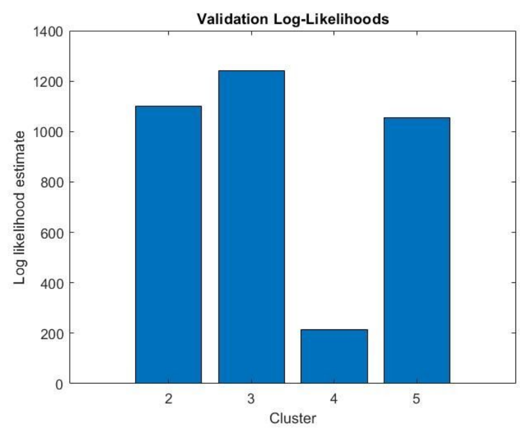

Figure 17. Percentage accuracy of a model trained with a Bird image from 2 to 5 clusters

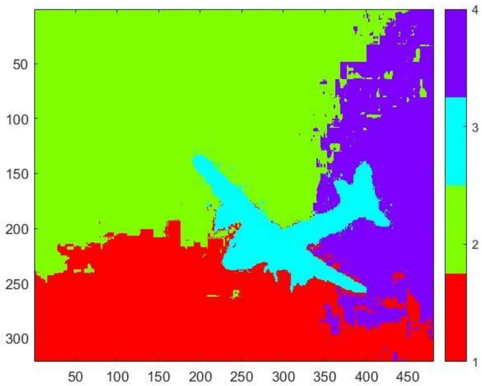

Figure 18. Plane color image segmented into the best number of clusters

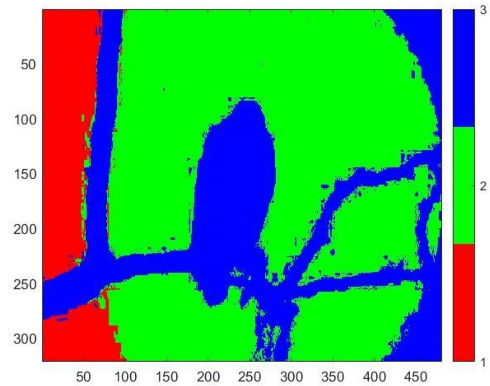

Figure 19. Bird color image segmented into the best number of clusters

Conclusion:

As we can see, the log-likelihood of the number of clusters of the plane image is relatively close. The 4 GMM clusters have the highest log-likelihood which might be a result of the blue sky, the cloud, and the darker cloud and the airplane. For the bird image, 3 GMM clusters have the highest log-likelihood which might be a result of the sky, the bird on the tree, and the shadow at the corner of the image.

Code: https://github.com/Alanbition/MachineLearning

Question1:

```
% Maximum likelihood training of a 2-layer MLP
% assuming additive (white) Gaussian noise
close all,
clear
dummyOut = 0;
% Input N specifies number of training samples

F = 10;
Perceptrons = 10;
numberOfClasses = 1;
numberOfSamples = 3;
numberOfDtrain_1000 = 1000;
numberOfDtest = 10000;


%Generate Dtrain and Dtest
Dtrain_1000 = exam4q1_generateData(numberOfDtrain_1000);
Dtest_10000 = exam4q1_generateData(numberOfDtest);

fig = 0;
fig = fig + 1;
figure(fig), plot(Dtrain_1000(1,:),Dtrain_1000(2,:),'.'),
xlabel('X_1'); ylabel('X_2');


fig = fig + 1;
figure(fig), plot(Dtest_10000(1,:),Dtest_10000(2,:),'.'),
xlabel('X_1'); ylabel('X_2');
%Use a for loop to iterate three datasets

numberOfDtrain = numberOfDtrain_1000;
Dtrain = Dtrain_1000(1,:);
Dtest = Dtest_10000(1,:);

Ylabels= Dtrain_1000(2,:);
YtestLabels = Dtest_10000(2,:);

%Split the data set to 10 block for 10 fold
block = ceil(linspace(0, numberOfDtrain, F+1));

for k = 1:F
   datasetBlock(k,:) = [block(k)+1,block(k+1)];
end
% Intialize likelihood
%
%
```

```
% ---------------------------------Start of Question 1-----------------------------------
numberOfaf = 3;

%Initialize array to store errors
errorTrain = zeros(F, Perceptrons);
errorValidate = zeros(F, Perceptrons);
errorAverage = zeros(3,Perceptrons);

MSE = zeros(F, Perceptrons);
MSEAverage = zeros(1, Perceptrons);
%Initialize a struct to store params result at the last 10fold operations
ParamsStore = struct();
FoldParamsStore = struct();
counter = 1;
 for nPerceptrons = 1:Perceptrons
        %Initializae params here so that next fold can use the previous
        %result(which result a better accuracy than put inside 10fold loop from my testing)

        counter = counter + 1;
        for k = 1:F

        %Assign validation and train set for each iteration
        validateIndex = [datasetBlock(k,1):datasetBlock(k,2)];
        if k == 1
            trainIndex = [datasetBlock(k, 2)+1:numberOfDtrain];
        elseif k == F
            trainIndex = [1:datasetBlock(k, 1)-1];
        else
            trainIndex = [1:datasetBlock(k-1, 2), datasetBlock(k+1, 2):numberOfDtrain];
        end

        %Select active function from sigmod, ISRU and SOFTPLUS

        type = "SOFTPLUS";


        yValidate = Ylabels(validateIndex);
        xValidate = Dtrain(validateIndex);
        validateLen = length(validateIndex);

        yTrain = Ylabels(trainIndex);
        xTrain = Dtrain(trainIndex);
        trainLen = length(trainIndex);

        X = xTrain;
        Y = yTrain;
        nX = size(X,1);
        nY = size(Y,1);
```

```
        params.A = randn(nPerceptrons,nX);
        params.b = randn(nPerceptrons,1);
        params.C = randn(nY,nPerceptrons);
        params.d = mean(Y,2);

        %disp(["xTrain", size(xTrain)])
        %Determine/specify sizes of parameter matrices/vectors

        sizeParams = [nX;nPerceptrons;nY];

        %Initialize model parameters
        %zeros(nY,1); % initialize to mean of y

        %params = paramsTrue;

        vecParamsInit = [params.A(:);params.b;params.C(:);params.d];

        %Optimize model
        options = optimset('MaxFunEvals',10000, 'MaxIter',10000); %Increase MaxFunEvals and MaxIter
        vecParams = fminsearch(@(vecParams)(objectiveFunction(type,
X,Y,sizeParams,vecParams)),vecParamsInit, options);


        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(type, xValidate,params);

        N = size(xValidate,2);
        MSE(k, nPerceptrons) = sum(sum((yValidate-H).*(yValidate-H),1),2)/N;

        %Calculate error percentage
        %[val, testIdx] = max(H);
        %[val, labelIdx] = max(yValidate);
        %error = find(testIdx~=labelIdx);
%           disp("H");
%           disp(H(1:10));
%           disp("yValidate");
%           disp(yValidate(1:10));
%           errorP = size(error)/size(xValidate);
%           errorTrain(k, nPerceptrons) = errorP;

        FoldParamsStore(k, nPerceptrons).A = params.A;
        FoldParamsStore(k, nPerceptrons).b = params.b;
```

```
        FoldParamsStore(k, nPerceptrons).C = params.C;
        FoldParamsStore(k, nPerceptrons).d = params.d;

    end
    disp([counter,"/11"])
    MSEAverage(1, nPerceptrons) = mean(MSE(:, nPerceptrons))
    %Calculate error Average
%     errorAverage(af, nPerceptrons) = mean(errorTrain(:, nPerceptrons));
%     errorTrain(:, nPerceptrons);
%     [val, minK] = min(errorTrain(:, nPerceptrons));
%     minK = min(minK(:))
    %Store best parms for fulture usage
%     ParamsStore(af,nPerceptrons).A = FoldParamsStore(minK, nPerceptrons).A;
%     ParamsStore(af,nPerceptrons).b = FoldParamsStore(minK, nPerceptrons).b;
%     ParamsStore(af,nPerceptrons).C = FoldParamsStore(minK, nPerceptrons).C;
%     ParamsStore(af,nPerceptrons).d = FoldParamsStore(minK, nPerceptrons).d;
    clear FoldParamsStore
 end


 MSEAverage

%  for nPerceptrons = 1:Perceptrons
%     for af = 1:numberOfaf
%         bar(1:Perceptrons, errorAverage(af, nPerceptrons))
%     end
%  end
 fig = fig + 1;
 figure(fig), clf,
 b1 = bar(1:nPerceptrons, MSEAverage),
 title("MSE for model trained with each perceptrons"),
 ylabel('MSE'),
 xlabel('Numbers of perceptrons'),
 legend('SOFTPLUS'),
 drawnow()

 [val, idx] = min(MSEAverage(:))
 nPerceptrons =find(MSEAverage==val)


 nPerceptrons = max(nPerceptrons(:))% In case same perceptrons performance


type = "SOFTPLUS";

X = Dtrain;%Y
Y = Ylabels;%training label
%Determine/specify sizes of parameter matrices/vectors
```

```
nX = size(X,1);
nY = size(Y,1);
sizeParams = [nX;nPerceptrons;nY];
%Initialize model parameters

params.A = randn(nPerceptrons,nX);
params.b = randn(nPerceptrons,1);
params.C = randn(nY,nPerceptrons);
params.d = mean(Y,2);

%Init with pervious best params
% params.A = ParamsStore(af,nPerceptrons).A;
% params.b = ParamsStore(af,nPerceptrons).b;
% params.C = ParamsStore(af,nPerceptrons).C;
% params.d = mean(Y,2);%ParamsStore(af,nPerceptrons).d;


vecParamsInit = [params.A(:);params.b;params.C(:);params.d];

%Optimize mode
options = optimset('MaxFunEvals',10000, 'MaxIter',10000);
vecParams = fminsearch(@(vecParams)(objectiveFunction(type,
X,Y,sizeParams,vecParams)),vecParamsInit, options);


%Visualize model output for training data
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(type, Dtest,params);
%Calculate error percentage
%[val, testIdx] = max(H);
%[val, labelIdx] = max(yValidate);
%error = find(testIdx~=labelIdx);

N = size(Dtest,2);
MSEFinal = sum(sum((YtestLabels-H).*(YtestLabels-H),1),2)/N
size(H)
fig = fig + 1;
figure(fig), plot(Dtest_10000(1,:),Dtest_10000(2,:),'.'),hold on
plot(Dtest_10000(1,:),H,'x')
xlabel('X_1'); ylabel('X_2');
legend('Original Data', 'Estimated Data'),
drawnow()
% error = find(H~=YtestLabels);
% errorP = size(error)/size(xValidate);
% disp(["Final Accuracy:", 1-errorP])
```

```
function objFncValue = objectiveFunction(type, X,Y,sizeParams,vecParams)
N = size(X,2); % number of samples
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(type, X,params);
objFncValue = sum(sum((Y-H).*(Y-H),1),2)/N;
%objFncValue = sum(-sum(Y.*log(H),1),2)/N;
% Change objective function to make this MLE for class posterior modeling
end

%
function H = mlpModel(type, X,params)
N = size(X,2);                          % number of samples
nY = length(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(type, 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 activations
%H = exp(V)./repmat(sum(exp(V),1),nY,1); % softmax nonlinearity for second/last layer
% Add softmax layer to make this a model for class posteriors
%
end

function out = activationFunction(type, in)
if type == "sigmod"
   out = 1./(1+exp(-in)); % logistic function
elseif type == "ISRU"
   out = in./sqrt(1+in.^2); % ISRU
else
   out = log(1+exp(in));% Soft Plus
end
end

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 [x1Grid,x2Grid,zGMM] = contourGMM(alpha,mu,Sigma,rangex1,rangex2)
x1Grid = linspace(floor(rangex1(1)),ceil(rangex1(2)),101);
x2Grid = linspace(floor(rangex2(1)),ceil(rangex2(2)),91);
[h,v] = meshgrid(x1Grid,x2Grid);
GMM = evalGMM([h(:)';v(:)'],alpha, mu, Sigma);
zGMM = reshape(GMM,91,101);
%figure(1),
contour(horizontalGrid,verticalGrid,discriminantScoreGrid,[minDSGV*[0.9,0.6,0.3],0,[0.3,0.6,0.9]*maxDS
GV]); % plot equilevel contours of the discriminant function
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 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
```

Question2:

```
% assuming additive (white) Gaussian noise
close all,
clear
dummyOut = 0;
% Input N specifies number of training samples

F = 10;
Perceptrons = 10;
numberOfClasses = 2;
numberOfDtrain_1000 = 1000;
numberOfDtest = 10000;


%Generate Dtrain and Dtest
[Dtrain_1000,DtrainLabels_1000] = generateMultiringDataset(numberOfClasses,numberOfDtrain_1000);
[Dtest,DtestLabels] = generateMultiringDataset(numberOfClasses,numberOfDtest);

%Plot Dtest
fig = 1
figure(fig), clf,
colors = rand(numberOfDtest,3);
for l = 1:numberOfClasses
   ind_l = find(DtestLabels==l);
   plot(Dtest(1,ind_l),Dtest(2,ind_l),'.','MarkerFaceColor',colors(l,:)), axis equal, hold on,
end
xlabel('x1'); ylabel('x2');
title(strcat('Sample data from Generated Dtest = ', num2str(numberOfDtest)));
drawnow()

%Plot Dtrain
fig = fig + 1;
figure(fig), clf,
colors = rand(numberOfDtest,3);
for l = 1:numberOfClasses
   ind_l = find(DtrainLabels_1000==l);
   plot(Dtrain_1000(1,ind_l),Dtrain_1000(2,ind_l),'.','MarkerFaceColor',colors(l,:)), axis equal, hold on,
end
xlabel('x1'); ylabel('x2');
title(strcat('Sample data from Generated Dtrain = ', num2str(numberOfDtrain_1000)));
drawnow()



for c=1:numberOfClasses
       index1 = find(DtestLabels==c);
       pTest(c) = size(index1,2)/numberOfDtest;
end
```

```
for c=1:numberOfClasses
        index2 = find(DtrainLabels_1000==c);
        pTrain(c) = size(index2,2)/numberOfDtrain_1000;
end

disp(pTest)
disp(pTrain)

DtrainLabels_1000 = DtrainLabels_1000-1;
l = 2*(DtrainLabels_1000-0.5);
x = Dtrain_1000;
N=1000; n = 2; K=10;

DtestLabels = DtestLabels-1;
lTest = 2*(DtestLabels-0.5);
xTest = Dtest;
% N=1000; n = 2; K=10;
% mu(:,1) = [-1;0]; mu(:,2) = [1;0];
% Sigma(:,:,1) = [2 0;0 1]; Sigma(:,:,2) = [1 0;0 4];
% p = [0.35,0.65]; % class priors for labels 0 and 1 respectively
% % Generate samples
% label = rand(1,N) >= p(1); l = 2*(label-0.5);
% Nc = [length(find(label==0)),length(find(label==1))]; % number of samples from each class
% x = zeros(n,N); % reserve space
% % Draw samples from each class pdf
% for lbl = 0:1
%     x(:,label==lbl) = randGaussian(Nc(lbl+1),mu(:,lbl+1),Sigma(:,:,lbl+1));
% end


% Train a Gaussian kernel SVM with cross-validation
% to select hyperparameters that minimize probability
% of error (i.e. maximize accuracy; 0-1 loss scenario)

dummy = ceil(linspace(0,N,K+1));
for k = 1:K, indPartitionLimits(k,:) = [dummy(k)+1,dummy(k+1)]; end,
CList = 10.^linspace(-1,9,11)
sigmaList = 10.^linspace(-2,3,13)

for sigmaCounter = 1:length(sigmaList)
    [sigmaCounter,length(sigmaList)],
    sigma = sigmaList(sigmaCounter);
    for CCounter = 1:length(CList)
        C = CList(CCounter);
        for k = 1:K
            indValidate = [indPartitionLimits(k,1):indPartitionLimits(k,2)];
            xValidate = x(:,indValidate); % Using folk k as validation set
```

```
            lValidate = l(indValidate);
            if k == 1
                indTrain = [indPartitionLimits(k,2)+1:N];
            elseif k == K
                indTrain = [1:indPartitionLimits(k,1)-1];
            else
                indTrain = [indPartitionLimits(k-1,2)+1:indPartitionLimits(k+1,1)-1];
            end
            % using all other folds as training set
            xTrain = x(:,indTrain); lTrain = l(indTrain);
            SVMk = fitcsvm(xTrain',lTrain,'BoxConstraint',C,'KernelFunction','RBF','KernelScale',sigma);
            dValidate = SVMk.predict(xValidate')'; % Labels of validation data using the trained SVM
            indCORRECT = find(lValidate.*dValidate == 1);
            Ncorrect(k)=length(indCORRECT);
        end
        PCorrect(CCounter,sigmaCounter)= sum(Ncorrect)/N;
    end
end

PCorrect(CCounter,sigmaCounter)

 fig = fig + 1;
 figure(fig), clf,
 b1 = bar(1:sigmaCounter, PCorrect),
 title('Probability of correct classification for model trained with each C and Sigma'),
 ylabel('Probability of correct classification'),
 xlabel('Sigma Value'),
  for iN = 1:length(CList)
      legendCell{iN} = num2str(CList(iN),'C = %e');
  end
  legend(legendCell),
 drawnow()


 fig = fig + 1;
 figure(fig), clf,
 b1 = bar(1:CCounter, PCorrect'),
 title('Probability of correct classification for model trained with each Sigma and C'),
 ylabel('Probability of correct classification'),
 xlabel('C Value'),
  for iN = 1:length(sigmaList)
      legendCell2{iN} = num2str(sigmaList(iN),'Sigma = %e');
  end
  legend(legendCell2),
 drawnow()


fig = fig + 1;
```

```
figure(fig), subplot(1,2,1),
contour(log10(CList),log10(sigmaList),PCorrect',20); xlabel('log_{10} C'), ylabel('log_{10} sigma'),
title('Gaussian-SVM Cross-Val Accuracy Estimate'), axis equal,
[dummy,indi] = max(PCorrect(:)); [indBestC, indBestSigma] = ind2sub(size(PCorrect),indi);
CBest= CList(indBestC)
sigmaBest= sigmaList(indBestSigma)
SVMBest = fitcsvm(x',l','BoxConstraint',CBest,'KernelFunction','RBF','KernelScale',sigmaBest);



d = SVMBest.predict(xTest')'; % Labels of training data using the trained SVM
indINCORRECT = find(lTest.*d == -1); % Find training samples that are incorrectly classified by the
trained SVM
indCORRECT = find(lTest.*d == 1); % Find training samples that are correctly classified by the trained
SVM
figure(fig), subplot(1,2,2),
plot(xTest(1,indCORRECT),xTest(2,indCORRECT),'g.'), hold on,
plot(xTest(1,indINCORRECT),xTest(2,indINCORRECT),'r.'), axis equal,
title('Testing Data (RED: Incorrectly Classified)'),
pTrainingError = length(indINCORRECT)/numberOfDtest, % Empirical estimate of training error
probability
Nx = 1001; Ny = 990; xGrid = linspace(-10,10,Nx); yGrid = linspace(-10,10,Ny);
[h,v] = meshgrid(xGrid,yGrid); dGrid = SVMBest.predict([h(:),v(:)]); zGrid = reshape(dGrid,Ny,Nx);
figure(fig), subplot(1,2,2), contour(xGrid,yGrid,zGrid,0); xlabel('x1'), ylabel('x2'), axis equal,
```

Question 3:

```
clear all, close all,


filenames{1,1} = '3096_color.jpg';
filenames{1,2} = '42049_color.jpg';

Kvalues = [2,3,4]; % desired numbers of clusters



for imageCounter = 1:2 %size(filenames,2)
    imdata = imread(filenames{1,imageCounter});
    figure(1), subplot(size(filenames,2),length(Kvalues)+1,(imageCounter-1)*(length(Kvalues)+1)+1),
imshow(imdata);
```

```
[R,C,D] = size(imdata); N = R*C; imdata = double(imdata);
rowIndices = [1:R]'*ones(1,C); colIndices = ones(R,1)*[1:C];
features = [rowIndices(:)';colIndices(:)']; % initialize with row and column indices
for d = 1:D
   imdatad = imdata(:,:,d); % pick one color at a time
   features = [features;imdatad(:)'];
end
minf = min(features,[],2); maxf = max(features,[],2);
ranges = maxf-minf;
x = diag(ranges.^(-1))*(features-repmat(minf,1,N)); % each feature normalized to the unit interval [0,1]
disp(size(x))




   %(1) 2 components
   x = x';
   GMM = fitgmdist(x, 2);
   disp(size(GMM));
   p = posterior(GMM, x);
   disp(size(p));
   [~, FirstImgIndx] = max(p, [], 2);
   disp(size(FirstImgIndx));
   figure
   lbls = reshape(FirstImgIndx,R, C);
   imagesc(lbls);
   colormap(hsv(2));
   colorbar('Ticks',1:2);


   x = x';
   %(2) 2-6 components
   %Split the data set to 10 block for 10 fold
   F = 10;
   block = ceil(linspace(0, size(x,2), F+1));

   for k = 1:F
      datasetBlock(k,:) = [block(k)+1,block(k+1)];
   end

   likelihoodValidate = zeros(F, 4);
   AverageValidate = zeros(1,4);


   for k = 1:F
         [row, col] = size(x);
         N = col;
         % Assign validation and train set for each iteration
```

```
        validateIndex = [datasetBlock(k,1):datasetBlock(k,2)];

        if k == 1
          trainIndex = [datasetBlock(k, 2)+1:N];
        elseif k == F
          trainIndex = [1:datasetBlock(k, 1)-1];
        else
          trainIndex = [1:datasetBlock(k-1, 2), datasetBlock(k+1, 2):N];
        end

        xValidate = [x(1,validateIndex);x(2, validateIndex);x(3, validateIndex);x(4, validateIndex);x(5,
validateIndex)];
        validateLen = length(validateIndex);

        xTrain = [x(1,trainIndex);x(2, trainIndex);x(3, trainIndex);x(4, trainIndex);x(5, trainIndex)];

        trainLen = length(trainIndex);
       for M = 2:5
          GMM = fitgmdist(xTrain', M);
          alpha = GMM.ComponentProportion;
          mu = (GMM.mu)';
          sigma = GMM.Sigma;
          likelihoodValidate(F,M-1) = sum(log(evalGMM(xValidate, alpha, mu, sigma)));
       end
   end

   AverageValidate = sum(likelihoodValidate)/F


   figure, clf,
   b1 = bar(2:5, AverageValidate),
   title('Validation Log-Likelihoods'),
   xlabel('Cluster'),
   ylabel(strcat('Log likelihood estimate')),
   drawnow()

   [val, BestIdx] = max(AverageValidate,[], 2)
   BestGMM = BestIdx+1

   x = x';
   GMM = fitgmdist(x, BestGMM);
   p = posterior(GMM, x);
   [~, newImgIndex] = max(p, [], 2);

   figure
   lbls = reshape(newImgIndex,R, C);
   imagesc(lbls);
   colormap(hsv(BestGMM));
```

```
    colorbar('Ticks',1:BestGMM);

    clear likelihoodValidate
    clear GMM
    clear xTrain
    clear xValidate
    clear datasetBlock
    clear AverageValidate
    clear newImgIndex
end




%
% Apply MAP to find which class has the highest probability result and
% compare with the test labels
% [val, testIdx] = max(GMMProbability);
% [val, labelIdx] = max(YtestLabels);
% error = find(testIdx~=labelIdx);
% errorP = size(error)/size(testIdx);
% AccuracyP(s) = 1-size(error)/size(testIdx);
%
%
%



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 [x1Grid,x2Grid,zGMM] = contourGMM(alpha,mu,Sigma,rangex1,rangex2)
```

```
x1Grid = linspace(floor(rangex1(1)),ceil(rangex1(2)),101);
x2Grid = linspace(floor(rangex2(1)),ceil(rangex2(2)),91);
[h,v] = meshgrid(x1Grid,x2Grid);
GMM = evalGMM([h(:)';v(:)'],alpha, mu, Sigma);
zGMM = reshape(GMM,91,101);
%figure(1),
contour(horizontalGrid,verticalGrid,discriminantScoreGrid,[minDSGV*[0.9,0.6,0.3],0,[0.3,0.6,0.9]*maxDS
GV]); % plot equilevel contours of the discriminant function
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 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
```