Michael Hodges

All files can be found here
https://github.com/Michael-Hodges/EECE5644_Machine_Learning.git
or in the appendix

Problem 1

Conduct the following model order selection exercise using 10-fold cross-validation procedure and report your procedure and results in a comprehensive, convincing, and rigorous fashion:

1. Select a Gaussian Mixture Model as the true probability density function for 2-dimensional real-valued data synthesis. This GMM will have 4 components with different mean vectors, different covariance matrices, and different probability for each Gaussian to be selected as the generator for each sample. Specify the true GMM that generates data.
2. Generate multiple data sets with independent identically distributed samples using this true GMM; these datasets will have respectively 10, 100, 1000, 10000 samples.
3. For each data set, using maximum likelihood parameter estimation principles (e.g. with the EM algorithm), within the framework of K(=10)-fold cross-validation, evaluate GMMs with different model orders; specifically evaluate candidate GMMs with 1, 2, 3, 4, 5, 6 Gaus- sian components. Note that both model parameter estimation and validation performance measures to be used is log-likelihood of data.
4. Report your results for the experiment, indicating which of the six GMM orders get selected for each of the datasets you produced. Develop a good way to describe and summarize your experiment results in the form of tables/figures.

Our true model is as follows:

Gaussian	Mean	Covariance	Prior
1	[2, 2]	[1 0; 0 1]	0.2
2	[-2, 2]	[0.1 0; 0 2]	0.3
3	[-2, -2]	[2 0; 0 0.1]	0.1
4	[2, -2]	[1 -0.7; -0.7 1]	0.4

The distribution is shown below:

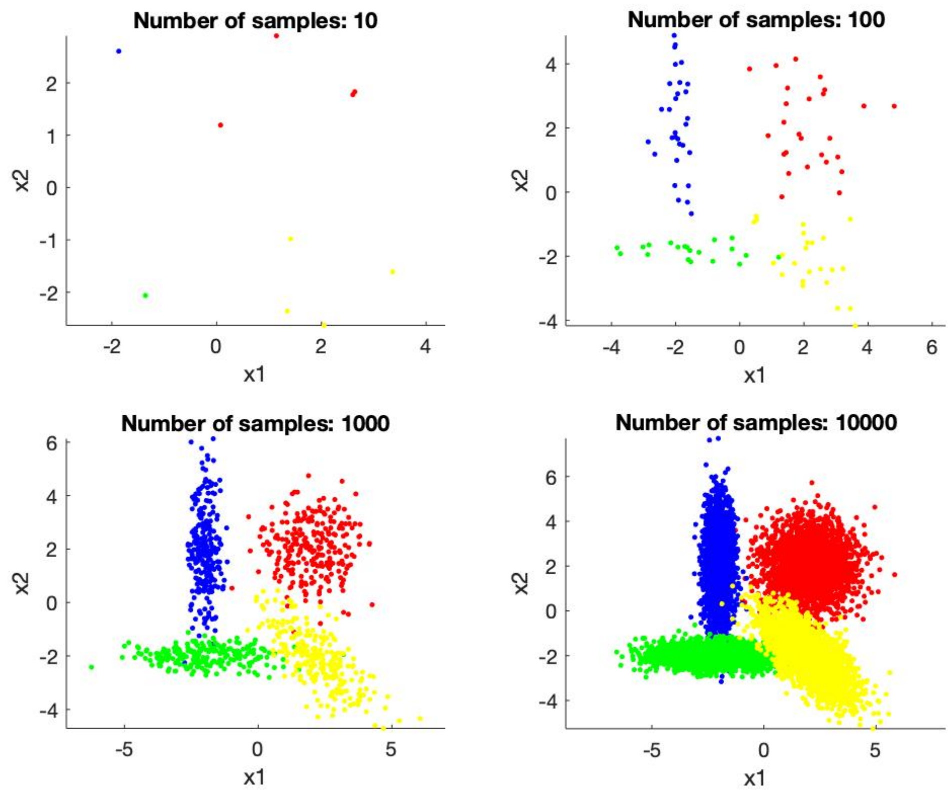

Figure 1: Original distribution for N= 10, 100, 1000, 10000

After performing k=10 cross fold validation and sweeping the number of mixture models to use from 1:6 this is our log loss distribution per fold shown in tables

-3.9638	-4.2978	-6.8071	-4.6400	-3.2910	-4.2495	-4.3731	-3.4920	-7.3494	-4.3322
-3.6247	-4.0536	-10.7713	-5.1538	-7.1021	-4.1788	-5.3541	-4.2197	-8.6037	-1.5662
-3.8892	-5.1617	-10.7713	-4.5627	-7.1024	-1.5734	-36.8864	-17.6312	-36.9400	-1.5674
-3.6245	-4.5059	-30.3148	-4.5627	-15.1959	-1.5734	-35.8748	-8.3713	-39.4311	-1.5674
-31.4103	-51.8002	-36.7608	-20.1127	-7.1024	-1.5734	-336.7273	-143.8673	-365.2600	-3.5226
-31.4103	-55.0988	-654.4114	-170.9504	-20.4518	0.8003	-186.4599	-349.1736	-575.3199	0.8003

Table 1: N = 10 Log loss with rows = number of gaussian models[1:6], columns = fold number [1:10]

-45.3176	-47.0058	-45.9258	-41.7257	-47.7805	-43.8224	-45.0922	-43.6180	-42.0198	-44.8303
-42.0365	-45.5029	-42.7794	-39.7459	-45.8585	-40.5367	-42.9307	-42.1857	-39.1282	-45.0525
-43.8706	-47.0594	-36.7414	-40.3071	-46.0681	-35.0121	-40.5685	-39.0395	-36.8316	-41.0783
-42.2338	-45.4390	-34.7023	-37.1690	-43.0764	-32.7697	-45.5257	-41.8924	-34.8293	-39.5333
-41.9808	-50.6697	-38.8393	-39.2670	-46.2515	-34.8718	-43.1371	-41.7510	-35.6455	-42.7909
-45.1952	-43.2287	-40.0771	-39.0436	-43.5951	-32.7287	-40.3351	-41.1926	-38.3550	-40.5447

Table 2: N = 100 Log loss with rows = number of gaussian models[1:6], columns = fold number [1:10]

-439.0841	-450.1371	-446.3358	-448.2369	-447.9725	-452.6688	-457.8052	-454.7543	-442.5781	-445.7355
-432.9024	-426.7419	-411.9246	-419.4427	-427.0980	-427.3741	-431.7883	-435.9189	-420.0560	-427.0808
-399.7574	-383.6188	-377.5681	-387.3891	-408.3147	-389.9040	-396.0273	-400.4866	-378.3312	-395.9905
-379.3314	-364.5988	-360.9133	-373.1415	-362.6514	-370.6030	-371.4420	-375.1668	-364.2285	-378.6182
-380.2124	-365.7982	-360.3902	-373.6248	-364.0981	-371.6876	-370.7591	-374.6762	-365.0757	-377.1173
-380.6687	-367.6231	-359.9976	-374.1620	-362.1270	-371.6685	-372.0235	-374.9097	-363.1726	-379.3920

Table 3: N = 1000 Log loss with rows = number of gaussian models[1:6], columns = fold number [1:10]

1.0e+03 *

-4.4492	-4.4438	-4.4639	-4.4081	-4.4572	-4.4560	-4.4625	-4.4154	-4.4546	-4.4613
-4.1879	-4.1647	-4.2326	-4.2342	-4.1896	-4.1636	-4.2086	-4.2095	-4.1851	-4.2288
-3.8551	-3.8175	-3.8071	-3.8415	-3.7987	-3.8481	-3.9486	-3.8723	-3.8624	-3.9101
-3.7209	-3.6626	-3.6680	-3.7086	-3.6493	-3.6613	-3.7310	-3.7159	-3.7051	-3.7123
-3.7220	-3.6620	-3.6683	-3.7089	-3.6498	-3.6620	-3.7315	-3.7158	-3.7048	-3.7122
-3.7210	-3.6624	-3.6690	-3.7088	-3.6496	-3.6617	-3.7308	-3.7157	-3.7056	-3.7121

Table 4: N = 10000 Log loss with rows = number of gaussian models[1:6], columns = fold number [1:10]

A graph summarizing the results is shown below:

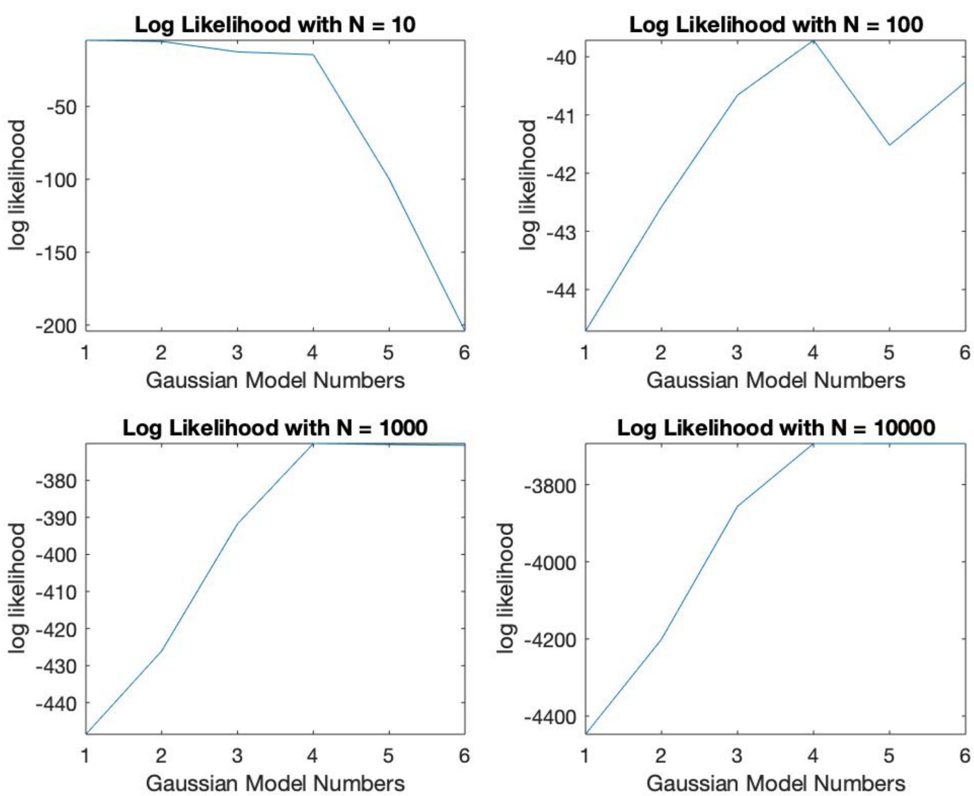

Figure 2: Log Likelihoods for N= 10, 100, 1000, 10000

As we can see after we reach a sample set of N = 100 our model chooses to have 4 Gaussian models in our Gaussian Mixture Model. At N = 10 there is not enough data to accurately find 4 models so our system chooses one that captures all of the data given the mean and covariance. Furthermore, for N = 1,000 and 10,000 there appears to about be a tie between 4-6 models, but we would choose the simplest model to represent our solution so we would choose 4 in this case.

Problem 2

Conduct the following maximum likelihood discriminative classifier training exercise on data generated from two Gaussian distributed classes:

1. Generate training set with 999 2-dimensional samples from two classes with priors $q_- = 0.3$ and $q_+ = 0.7$; the class-conditional data probability distributions are two Gaussians with different mean vectors and different covariance matrices (choose the matrices to be non- diagonal with distinct eigenvalues, so your Gaussian pdfs are tilted with respect to each other and elongated in different directions by different aspect ratios). Hint: For more interesting results, make the Gaussians overlap with each other somewhat significantly, so that the minimum error probability achievable is not too small.

2. Using Fisher LDA, identify a linear classifier that minimizes the error count on the training set. This classifier will have a discriminant function of the form wTLDAx + bLDA where x is the data vector, and the classifier decides in favor of Class - if the discriminant is below 0, and decides in favor of Class +, if the discriminant is at least 0.

3. Train the parameters of a logistic function $y(x) = 1/(1+e(w^T x+b))$ using the maximum likelihood estimation principle to optimize the parameters w and b with the training set, such that the function is trained to act as a surrogate for the posterior probability of Class + given x. In particular, your model assumes that $y(x) = P(Label = +|x)$; consequently, $1-y(x) = P(Label = -|x)$. Hint: Once you specify the optimization objective to train this logistic-linear-model for class posterior, you can solve the optimization problem using any suitable numerical optimization procedure, such as gradient ascent that you implement from scratch, or using a derivative free numerical optimization procedure like the Nelder-Mead Simplex Reflection Algorithm (e.g. in Matlab, fminsearch). Make a choice, implement correctly, perhaps consider using the LDA solution you developed earlier to provide an initial estimate for the model parameters.

4. Report visual and numerical results that compare the following three classifiers (e.g. data scatter plots with color/shape indicators of true/decided labels), including the error counts each classifier achieve on the training set: MAP-classifier that makes use of the true data distributions and class priors, which achieves minimum probability error by design; LDA classifier you designed earlier; logistic-linear classifier you designed next.

$$
\begin{aligned}
\hat{\theta}_{ML} &= \underset{\Theta}{\operatorname{argmax}} \sum_{i=1}^{N} \ln P(x_i, l_i|\Theta) \\
&= \underset{\Theta}{\operatorname{argmax}} \sum_{i=1}^{N} \ln \left[(P(l_i|x_i, \Theta) P(x_i|\Theta) \right] \\
&= \underset{\Theta}{\operatorname{argmax}} \sum_{i=1}^{N} \ln \left[(P(l_i|x_i, \Theta) P(x_i|\Theta) \right] \\
&= \underset{\Theta}{\operatorname{argmax}} \sum_{i=1}^{N} \ln \left(P(l_i|x_i, \Theta) + \sum_{i=1}^{N} \ln P(x_i|\Theta) \right. \\
&= \underset{\Theta}{\operatorname{argmax}} \sum_{i=1}^{N} \ln y(x)^{q_i} (1 - y(x))^{1-q_i} \\
&= \underset{\Theta}{\operatorname{argmax}} \sum_{i=1}^{N} q_i \ln y(x) + (1 - q_i) \ln(1 - y(x)) \\
&= \underset{\mathbf{w}, b}{\operatorname{argmax}} \sum_{i=1}^{N} q_i \ln \left(\frac{1}{1 + e^{(\mathbf{w}^T \mathbf{x} + b)}} \right) + (1 - q_i) \ln \left(1 - \frac{1}{1 + e^{(\mathbf{w}^T \mathbf{x} + b)}} \right)
\end{aligned}
$$

Where q_i is the prior probability for the class x belongs to. Note: we essentially use a bernoulli to solve this.

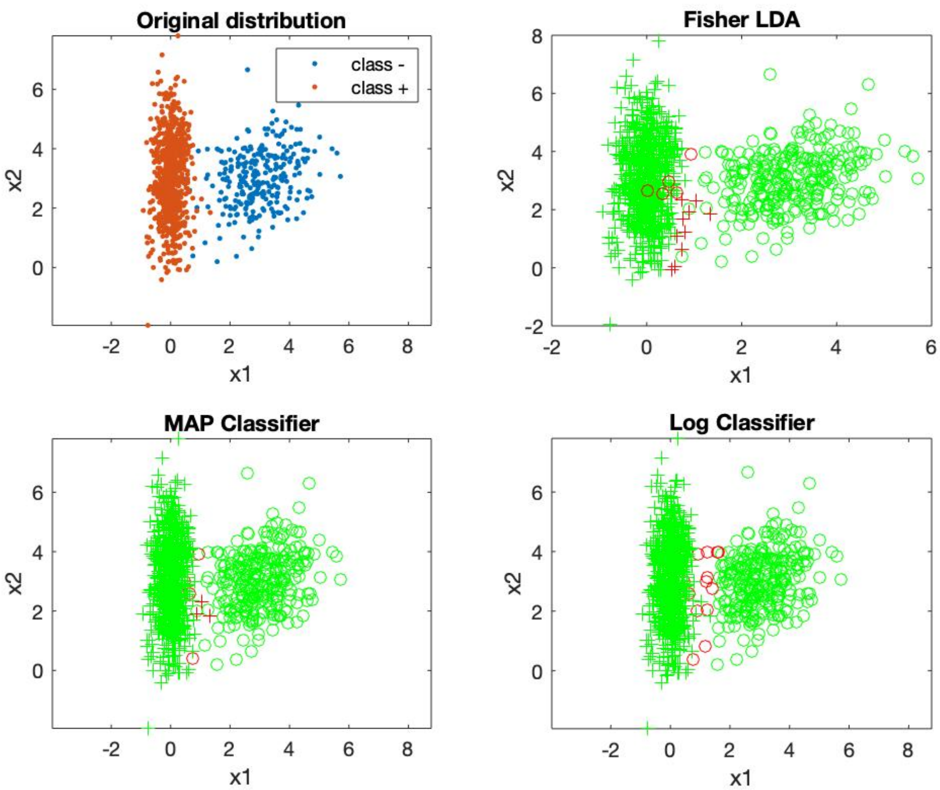

Figure 3: True class labels and classification. 0 class -, + Class +, Red incorrect, green correct

FischerLDA	MAP	Log
1.5%	0.9%	1.5%

Table 5: Percent mislabeled per model

FischerLDA	MAP	Log
15	9	15

Table 6: Number mislabeled per model

Appendix

1. Problem 1

```
clear all; close all; clc
% Gaussian Mixture Model Parameters
% Means:
mu(:,1) = [2, 2];
mu(:,2) = [-2, 2];
mu(:,3) = [-2, -2];
mu(:,4) = [2, -2];

% Covariances:
Sigma(:,:,1) = [1 0; 0 1];
Sigma(:,:,2) = [0.1 0; 0 2];
Sigma(:,:,3) = [2 0; 0 0.1];
Sigma(:,:,4) = [1 -0.7; -0.7 1];

% Priors:
classPriors = [0.2 0.3 0.1 0.4]; % True Priors
classPriors1 = [0.25 0.25 0.25 0.251];
assert(max(cumsum(classPriors)) == 1, 'Priors do not equal 1');
thr = [0,cumsum(classPriors1)];

disp("True Means");
disp(mu);
disp("True Covariances:");
disp(Sigma);
disp("True Priors:");
disp(classPriors);
k = 10; % Perform k fold validation

%Generate Data:
% figure(1),clf, colorList = 'rbgy';
for i = 1:4
    N = 10^i; % Number of samples to generate for each set (10,100,1000,10000)
    u = rand(1,N); L = zeros(1,N); x = zeros(2,N);
    for l = 1:4
        indices = find(thr(l)<=u & u<thr(l+1)); % fixed using classPriors1 adding a small term to last prior 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(mu(:,l),Sigma(:,:,l),length(indices))';
%           subplot(2,2,i);
%           plot(x(1,indices),x(2,indices),'.','MarkerFaceColor',
```

```
    colorList(l)); axis equal, hold on,
%           axis([-10 10 -10 10]);
    end
%       hold off;
    assert(isempty(find(L==0, 1)),'some values unclassified');
    x(3,:) = L;
    disp("Number Samples N = " + 10^i);
    disp("Class1: " + num2str(length(find(L==1))));
    disp("Class2: " + num2str(length(find(L==2))));
    disp("Class3: " + num2str(length(find(L==3))));
    disp("Class4: " + num2str(length(find(L==4))));


        % Loop through number of models to try 1-6
    regWeight = 1e-10; % regularization parameter for covariance estimates
    for l = 1:6 % number of models to loop through
        disp("performing: " + k + "-fold cross validation N=" + 10^i + ", Models =" + l);
        for q = 1:k % k fold cross validation
            disp(q+"th fold");
            dVal = x(:,1+(N/k)*(q-1):N/k*q);
%               disp("All " + num2str(x));
%               disp("Validation: " + num2str(dVal));
            dTrain = x;
            dTrain(:,1+(N/k)*(q-1):N/k*q) = [];
%               disp("Train: " + num2str(dTrain));
            % Set up initial GMM for l number of Gaussians
            EMdTrain = dTrain; % take train and set equal to new set to work with
            shuffledIndices = randperm(length(EMdTrain(1,:))); % Pick l random samples to be initial mean
            EM_mu = EMdTrain(1:2,shuffledIndices(1:l)); EM_mu_new = EM_mu; % starting means
%               [~,EMdTrain(3,:)] = min(pdist2(EM_mu',EMdTrain(1:2,:)'),[],1); % Set labels to nearest centroid
%               EM_Sigma = zeros(2,2,l); EM_Sigma_new = EM_Sigma;
%               EM_Prior = zeros(1,l); EM_Prior_new = EM_Prior;
            % initialize Priors
%               EM_Prior = ones(1,l)./l;
%               for b = 1:l
%                   EM_Prior(b) = length(find(EMdTrain(3,:)==b))/length(EMdTrain(3,:));
%               end
            % Initialize Covariances
%               for b = 1:l
```

```
%                       EM_Sigma(:,:,b) = cov(EMdTrain(1:2,find(EMdTrain(3,:)==b))') + regWeight*eye(2,2);% Generate covariances with given labels
%                   end
%                   epsilon = 1e-4; % convergence factor
%                   Converged=0;
%                   t = 0;
                % Try two initializations
                [EMlog1, EM_Prior1, EM_mu1, EM_Sigma1] = EMforGMM_Edited(length(EMdTrain(3,:)), EMdTrain(1:2,:), l);
                [EMlog2, EM_Prior2, EM_mu2, EM_Sigma2] = EMforGMM_Edited(length(EMdTrain(3,:)), EMdTrain(1:2,:), l);
                if EMlog1>EMlog2
                    EM_Prior = EM_Prior1;
                    EM_mu = EM_mu1;
                    EM_Sigma = EM_Sigma1;
                else
                    EM_Prior = EM_Prior2;
                    EM_mu = EM_mu2;
                    EM_Sigma = EM_Sigma2;
                end
                loglikelihood(l,q,i) = sum(log(evalGMM(dVal(1:2,:),EM_Prior,EM_mu,EM_Sigma)));
            end
            disp(loglikelihood(:,:,i));
            %Store error here for each model

        end
        figure(2); hold on;
        subplot(2,2,i);
        means = mean(loglikelihood(:,:,i),2);
        plot(1:6,means');
        xlabel("Gaussian Model Numbers");
        ylabel("log likelihood");
        title("Log Likelihood with N = " + 10^i);
        axis([1 6 -inf inf]);
        rangex1 = [min(x(1,:)),max(x(1,:))];
        rangex2 = [min(x(2,:)),max(x(2,:))];
        [x1Grid,x2Grid,zGMM] = contourGMM(EM_Prior,EM_mu,EM_Sigma,rangex1,rangex2);
        figure(3);
        subplot(2,2,i); hold on;
        plot(x(1,find(L==1)),x(2,find(L==1)),'.r');
        plot(x(1,find(L==2)),x(2,find(L==2)),'.b');
```

```
        plot(x(1,find(L==3)),x(2,find(L==3)),'.g');
        plot(x(1,find(L==4)),x(2,find(L==4)),'.y');
%       contour(x1Grid,x2Grid,zGMM);
        title("Number of samples: " + 10^i);
        xlabel("x1");
        ylabel("x2");
        axis('equal');
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

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);
end
```

2. Problem 1 EM

```
function [logLikelihood, alpha,mu,Sigma] = EMforGMM_Edited(N,data, parameters)
% Generates N samples from a specified GMM,
% then uses EM algorithm to estimate the parameters
% of a GMM that has the same number of components
% as the true GMM that generates the samples.

% close all,
delta = 1e-2; % tolerance for EM stopping criterion
regWeight = 5e-3; % regularization parameter for covariance
```

```
estimates

% Generate samples from a 3-component GMM
% alpha_true = [0.2,0.3,0.5];
% mu_true = [-10 0 10;0 0 0];
% Sigma_true(:,:,1) = [3 1;1 20];
% Sigma_true(:,:,2) = [7 1;1 2];
% Sigma_true(:,:,3) = [4 1;1 16];
% x = randGMM(N,alpha_true,mu_true,Sigma_true);
% [d,M] = size(mu_true); % determine dimensionality of samples and number of GMM components
d=2;
M = parameters;
x = data;
% Initialize the GMM to randomly selected samples
alpha = ones(1,M)/M;
shuffledIndices = randperm(N);
mu = x(:,shuffledIndices(1:M)); % pick M random samples as initial mean estimates
[~,assignedCentroidLabels] = min(pdist2(mu',x'),[],1); % assign each sample to the nearest mean
for m = 1:M % use sample covariances of initial assignments as initial covariance estimates
    Sigma(:,:,m) = cov(x(:,find(assignedCentroidLabels==m))') + regWeight*eye(d,d);
end
t = 0; %displayProgress(t,x,alpha,mu,Sigma);

Converged = 0; % Not converged at the beginning
while ~Converged
    for l = 1:M % multiply prior with PDF evaluated at certain point
        temp(l,:) = repmat(alpha(l),1,N).*evalGaussian(x,mu(:,l),Sigma(:,:,l));
    end
    plgivenx = temp./sum(temp,1); % Probability of each given data
    alphaNew = mean(plgivenx,2); % New Priors
    w = plgivenx./repmat(sum(plgivenx,2),1,N); % Probability of given data normalized
    muNew = x*w'; % update means mutliplying data by prior distributions
    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(sum(abs(alphaNew-alpha)));
    Dmu = sum(sum(abs(muNew-mu)));
    DSigma = sum(sum(sum(abs(abs(SigmaNew-Sigma)))));
    alpha = alphaNew; mu = muNew; Sigma = SigmaNew;
    Converged = ((Dalpha+Dmu+DSigma)<delta); % Check if converged
    t = t+1;
%     displayProgress(t,x,alpha,mu,Sigma);
end
logLikelihood = sum(log(evalGMM(x,alpha,mu,Sigma)));
%keyboard,

%%%
function displayProgress(t,x,alpha,mu,Sigma)
figure(1),
if size(x,1)==2
    subplot(1,2,1), cla,
    plot(x(1,:),x(2,:),'b.');
    xlabel('x_1'), ylabel('x_2'), title('Data and Estimated GMM Contours'),
    axis equal, hold on;
    rangex1 = [min(x(1,:)),max(x(1,:))];
    rangex2 = [min(x(2,:)),max(x(2,:))];
    [x1Grid,x2Grid,zGMM] = contourGMM(alpha,mu,Sigma,rangex1,rangex2);
    contour(x1Grid,x2Grid,zGMM); axis equal,
    subplot(1,2,2),
end
logLikelihood = sum(log(evalGMM(x,alpha,mu,Sigma)));
plot(t,logLikelihood,'b.'); hold on,
xlabel('Iteration Index'), ylabel('Log-Likelihood of Data'),
drawnow; pause(0.1),

%%%
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

%%%
```

```
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);

%%%
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]*maxDSGV]); % plot equilevel contours of the discriminant function

%%%
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

%%%
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);
```

3. Problem 2

```
clear all; close all; clc
% Gaussian Mixture Model Parameters
% Means:
mu(:,1) = [3, 3]; % Q-
mu(:,2) = [0, 3]; %  Q+

% Covariances:
```

```
Sigma(:,:,1) = [1 .4; .4 1]; % Q-
Sigma(:,:,2) = [0.1 0; 0 2]; % Q+

% Priors:
classPriors = [0.3 0.7]; % True Priors
classPriors1 = [0.3 0.7];
assert(max(cumsum(classPriors)) == 1, 'Priors do not equal 1');
thr = [0,cumsum(classPriors1)];

N = 999; % Number of Samples to take
% Generate samples from each class
figure(1),clf, colorList = 'rbg';
subplot(2,2,1);
u = rand(1,N);
for l = 1:2
    indices = find(thr(l)<=u & u<thr(l+1)); % fixed using
       classPriors1 adding a small term to last prior 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(mu(:,l),Sigma(:,:,l),length(indices))
       ';
    plot(x(1,indices),x(2,indices),'.','MarkerFaceColor',
       colorList(l)); axis equal, hold on,
    axis('equal');
end
legend('class -', 'class +');
title('Original distribution');
xlabel('x1'), ylabel('x2');
assert(isempty(find(L==0, 1)),'some values unclassified');
x(3,:) = L;
disp("Number Samples N = " + N);
disp("Class1: " + num2str(length(find(L==1))));
disp("Class2: " + num2str(length(find(L==2))));

mu1hat = mean(x(find(L==1))); S1hat = cov(x(find(L==1)));
mu2hat = mean(x(find(L==1))); S2hat = cov(x(find(L==1)));

% Sb = (mu1hat-mu2hat)*(mu1hat-mu2hat)';
% Sw = S1hat + S2hat;
Sb = (mu(:,1)-mu(:,2))*(mu(:,1)-mu(:,2))';
Sw = Sigma(:,:,1) + Sigma(:,:,2);

[V,D] = eig(inv(Sw)*Sb);
[~,ind] = sort(diag(D),'descend');
wLDA = V(:,ind(1)); % Fisher LDA projection vector
```

```
disp(wLDA)
yLDA = wLDA'*x(1:2,:); % All data projected on to the line spanned by wLDA
wLDA = sign(mean(yLDA(find(L==2)))-mean(yLDA(find(L==1))))*wLDA; % ensures class1 falls on the + side of the axis
yLDA = sign(mean(yLDA(find(L==2)))-mean(yLDA(find(L==1))))*yLDA; % flip yLDA accordingly
disp(wLDA)
% figure(2), clf,
% subplot(2,2,2);
% plot(yLDA(find(L==1)),zeros(1,length(find(L==1))),'o'), hold on,
% plot(yLDA(find(L==2)),zeros(1,length(find(L==2))),'+'), axis equal,
% legend('Class -','Class +'),
% title('LDA projection of data and their true labels'),
% xlabel('x_1'), ylabel('x_2'),
tau = 0;
decisionLDA = (yLDA >= -classPriors(1)/classPriors(2)); % classPriors(2)/classPriors(1)
decisionLDA = decisionLDA+1;

%Note: wLDA is w and classPriors(1)/classPriors(2) is b

ind00LDA = find(L==1 & decisionLDA==1);
ind01LDA = find(L==1 & decisionLDA==2);
ind10LDA = find(L==2 & decisionLDA==1);
ind11LDA = find(L==2 & decisionLDA==2);

disp((length(ind01LDA)+length(ind10LDA))/N);
% figure(3), clf,
% plot(yLDA(ind00),zeros(1,length(yLDA(ind00))),'og'), hold on,
% plot(yLDA(ind01),zeros(1,length(yLDA(ind01))),'or');
% plot(yLDA(ind11),zeros(1,length(yLDA(ind11))),'+g');
% plot(yLDA(ind10),zeros(1,length(yLDA(ind10))),'+r');
% legend('class - correct', 'class 0- incorrect', 'class + correct', 'class + incorrect')
% figure(4), clf,
subplot(2,2,2);
plot(x(1,ind00LDA),x(2,ind00LDA),'og'), hold on,
plot(x(1,ind11LDA),x(2,ind11LDA),'+g');
plot(x(1,ind01LDA),x(2,ind01LDA),'or');
plot(x(1,ind10LDA),x(2,ind10LDA),'+r');
title('Fisher LDA');
xlabel('x1'),ylabel('x2');
```

```
%hleg = legend('class - correct', 'class + correct', 'class -
    incorrect', 'class + incorrect');
%set(hleg,'fontsize',14)


%MAP

lambda = [0 1;1 0]; % loss values
gamma = (lambda(2,1)-lambda(1,1))/(lambda(1,2)-lambda(2,2)) *
    classPriors(1)/classPriors(2); %threshold
discriminantScore = log(mvnpdf(x(1:2,:)',mu(:,2)',Sigma(:,:,2))
    )-log(mvnpdf(x(1:2,:)',mu(:,1)',Sigma(:,:,1)));% - log(gamma
    );
decision = (discriminantScore >= log(gamma));
decision = decision+1;

ind00MAP = find(decision'==1 & L==1); %p00 = length(ind00)/Nc
    (1); % probability of true negative
ind10MAP = find(decision'==2 & L==1); %p10 = length(ind10)/Nc
    (1); % probability of false positive
ind01MAP = find(decision'==1 & L==2); %p01 = length(ind01)/Nc
    (2); % probability of false negative
ind11MAP = find(decision'==2 & L==2); %p11 = length(ind11)/Nc
    (2); % probability of true positive
%p(error) = [p10,p01]*Nc'/N; % probability of error,
    empirically estimated

% figure(5), % class 0 circle, class 1 +, correct green,
    incorrect red
subplot(2,2,3);
plot(x(1,ind00MAP),x(2,ind00MAP),'og'); hold on,
plot(x(1,ind10MAP),x(2,ind10MAP),'or'); hold on,
plot(x(1,ind01MAP),x(2,ind01MAP),'+r'); hold on,
plot(x(1,ind11MAP),x(2,ind11MAP),'+g'); hold on,
title('MAP Classifier');
xlabel('x1'); ylabel('x2');
axis equal,

disp((length(ind01MAP)+length(ind10MAP))/N);

% Logarithmic classifier
% fun = @(w)for 1+exp(w(1)'*x(1:2,:)+w(2,1));
x(3,:) = 1;
w0 = [1 1 1];
%fun = @(w,x)sum(-classPriors(1)*log(1./(1+exp(w(1:2)*x(1:2,:)+
    w(3))))-classPriors(2)*log(1-1./(1+exp(w(1:2)*x(1:2,:)+w(3))
```

```
)));
fun = @(w,x,L)sum( (-classPriors(L).*log(1./(1+exp(w*x)))) - ((1-classPriors(L)).*log(1-(1./(1+exp(w*x))))));
%fun = @(w,x)-sum(exp(w(1:2)*x(1:2,find(L==1))-w(3)))+sum(exp(w(1:2)*x(1:2,find(L==2))-w(3)));
f = @(w)fun(w,x,L);
w_l = fminsearch(f,w0);

logDscore = 1./(1+exp(w_l*x));
logDecision = (logDscore>=0.5);
logDecision = logDecision+1;

ind00log = find(logDecision==1 & L==1); %p00 = length(ind00)/Nc(1); % probability of true negative
ind10log = find(logDecision==2 & L==1); %p10 = length(ind10)/Nc(1); % probability of false positive
ind01log = find(logDecision==1 & L==2); %p01 = length(ind01)/Nc(2); % probability of false negative
ind11log = find(logDecision==2 & L==2); %p11 = length(ind11)/Nc(2); % probability of true positive

subplot(2,2,4);
plot(x(1,ind00log),x(2,ind00log),'og'); hold on,
plot(x(1,ind10log),x(2,ind10log),'or'); hold on,
plot(x(1,ind01log),x(2,ind01log),'+r'); hold on,
plot(x(1,ind11log),x(2,ind11log),'+g'); hold on,
title('Log Classifier');
xlabel('x1'); ylabel('x2');
axis equal,
```