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

Problem 1

Using the K-Means clustering algorithm with minimum Euclidean-distance-based assignments of samples to cluster centroids, segment the two attached color images into $K \in 2, 3, 4, 5$ segments. As the feature vector for each pixel use a 5-dimensional feature vector consisting of normalized vertical and horizontal coordinates of the pixel relative to the top-left corner of the image, as well as normalized red, green, and blue values of the image color at that pixel. Normalize value to make best use of the range of gray values at your disposal for visualization.

For each $K \in 2, 3, 4, 5$, let the algorithm assign labels to each pixel; specifically, label $l_r c \in 1, ..., K$ to the pixel located at row r and column c. Present your clustering results in the form of an image of these label values. Make sure you improve this segmentation outcome visualization by using a contrast enhancement method; for instance, assign a unique color value to each label and make your label image colored, or assign visually distinct grayscale value levels to each label value to make best use of the range of gray values at your disposal for visualization.

Repeat this segmentation exercise using GMM-based clustering. For each specific K, use the EM algorithm to fit a GMM with K components, and then use that GMM to do MAP-classification style cluster label assignments to pixels. Display results similarly for this alternative clustering method. Briefly comment on the reasons of any differences, if any.

Figure 1: Original

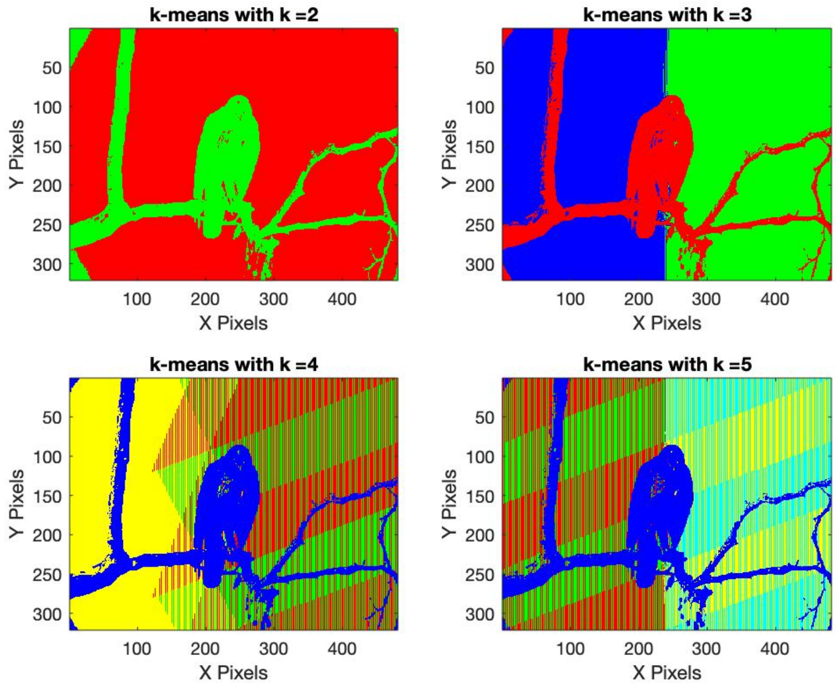


Figure 2: K-Means

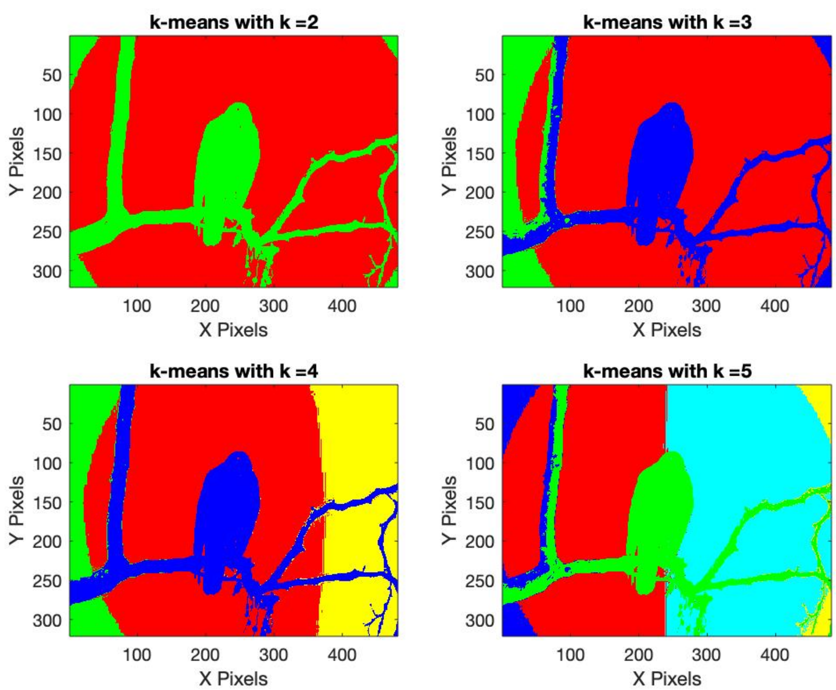


Figure 3: GMM

Figure 4: Original

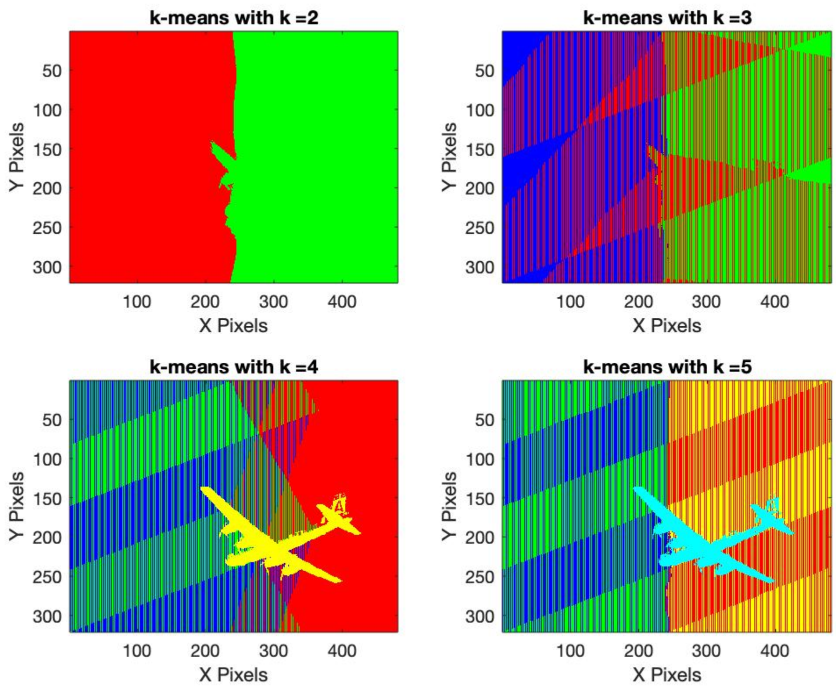


Figure 5: K-Means

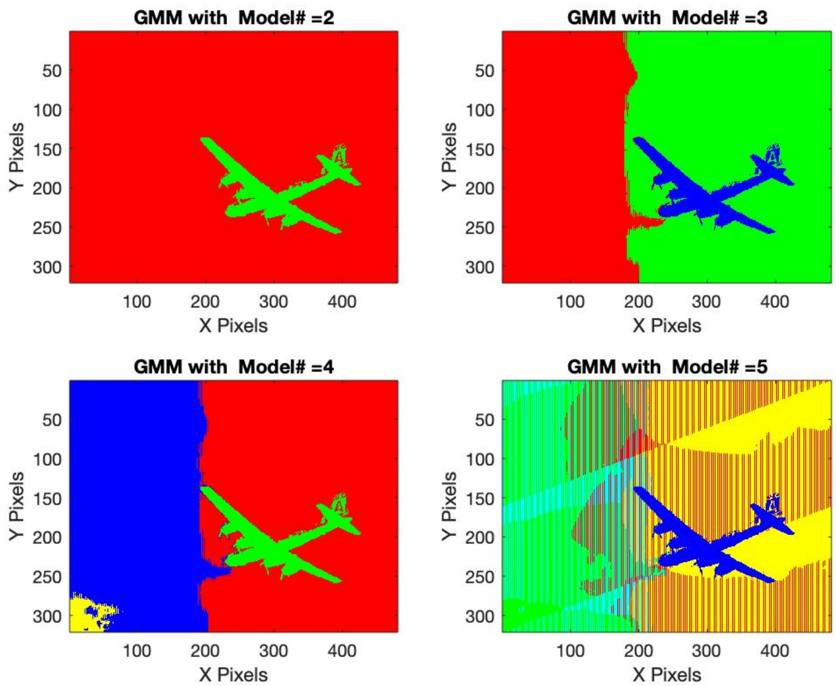


Figure 6: GMM

As we can see the GMM generally performs better. The main reasoning is that K-Means gives a linear boundary, and based on the features we use we might not be able accurately draw a linear boundary. As we can see once we start grouping with a certain number of clusters the algorithm begins to struggle with the background of the image. Overall, we are able to pick out the main target (e.g. bird on stick, and plane) using these clustering algorithms.

Problem 2

In this exercise, you will train two support vector machine (SVM) classifiers and assess/-compare their test performances. These SVMs wil respectively have linear and spherically-symmetric Gaussian (shaped radial basis function) kernels. We will refer to them as Linear-SVM and Gaussian-SVM. The data vectors are two-dimensional real-valued. The data distributions for the two classes are as follows: (1) data from class -1 are drawn from a Gaussian with zero-mean and identity- covariance-matrix; (2) data from class +1 are generated using a two-step procedure: a radius value is drawn from a uniform distribution over the interval $[2, 3]$ and an angle value (in radians) is drawn from a uniform distribution over the interval $[-\pi, \pi]$; these radius and angle values are converted to Cartesian coordinates using the Polar-to-Cartesian coordinate transformation rule.

1. Generate a training set with 1000 independent samples from these two class distributions with priors $q_- = 0.35$ and $q_+ = 0.65$; note that this does not mean 350 samples from one class and 650 from the other - the class label needs to be randomly selected for each sample, in accordance with this prior. Visualize your training data.

2. Using 10-fold cross-validation, and minimum probability of error as the objective, select the hyper parameters for both Linear-SVM and Gaussian-SVM. For both classifiers, the constraint violation term weight (usually denoted by C; sometimes called the overlap penalty weight; referred to as the box constraint parameter in Matlab's fitcsvm) must be optimized. For the Gaussian kernel, the scale parameter (usually denoted by σ , corresponds to the standard deviation, if this Gaussian was a probability distribution) needs to be optimized. Visualize your cross-validation process in search of optimal hyperparameter values. Report the smallest probability of error estimate you get from cross-validation.

3. Using the best hyperparameters you identified, train your Linear-SVM and Gaussian-SVM using all of the training dataset. Visualize classification results on training data, count the erroneously classified samples and report the training dataset probability of error estimate.

4. Generate 1000 independent test samples from the same class distributions with the same priors as in the training dataset. Apply the Linear-SVM and Gaussian-SVM classifiers to the test data samples. Visualize the performance of your classifiers on the test dataset and report your test probability of error estimate.

In our test we will sweep over 13 values of C, the overlap penalty weight, for the linear SVM. For the gaussian SVM we will sweep over 23 values for C, the overlap penalty, and σ, the standard deviation.

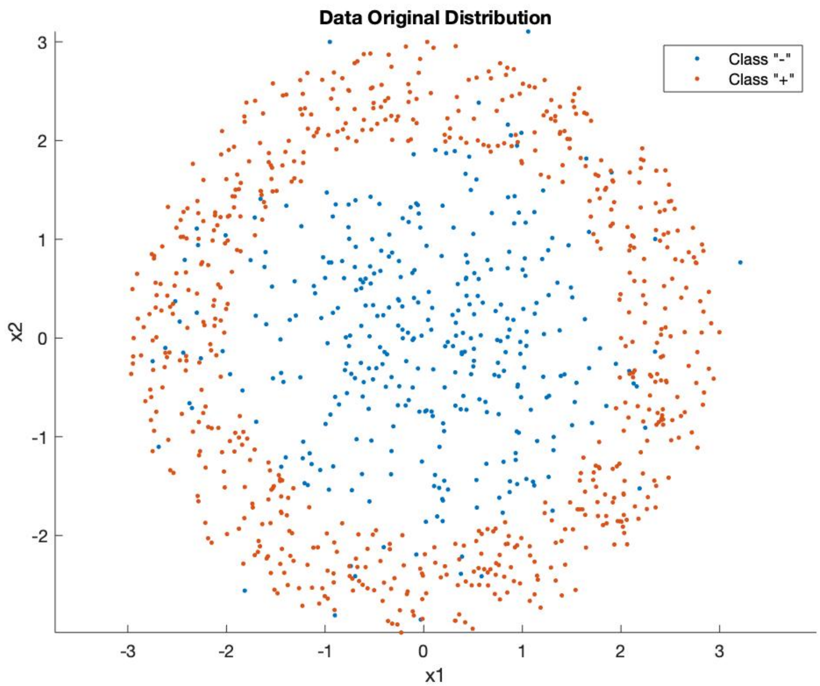
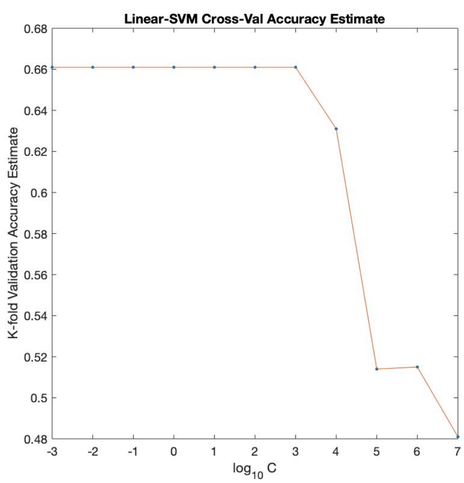


Figure 7: Train Data Original Distribution



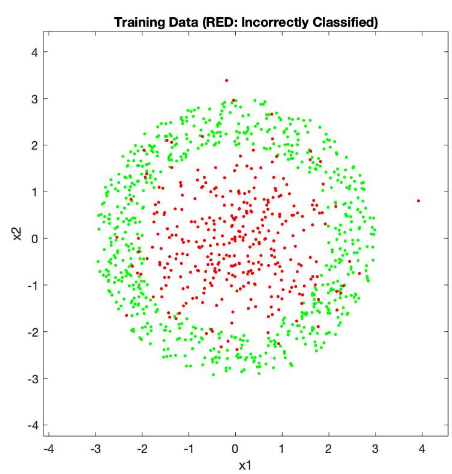


Figure 8: Train data with Linear SVM with best parameters

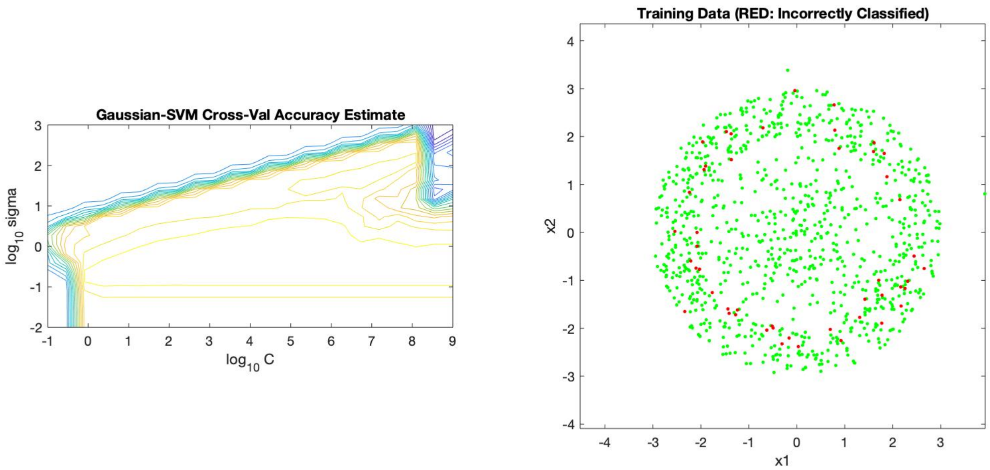


Figure 9: Train data Gaussain SVM with best parameters

From the training data we receive the following errors: Smallest Probability of error from cross validation

Linear Train Error: 33.50%

Gaussian Train Error: 4.3%

Probability of error for all data

Linear Train Error: 33.90%

Gaussian Train Error: 4.8%

This makes sense as the linear SVM shouldn't work since the data cannot be linearly seperated. Thus to reduce errror we classify all of the classes as one and hence choose the class with the greatest prior. The results show what we would expect with around 35% error which matches the prior for the less likely class.

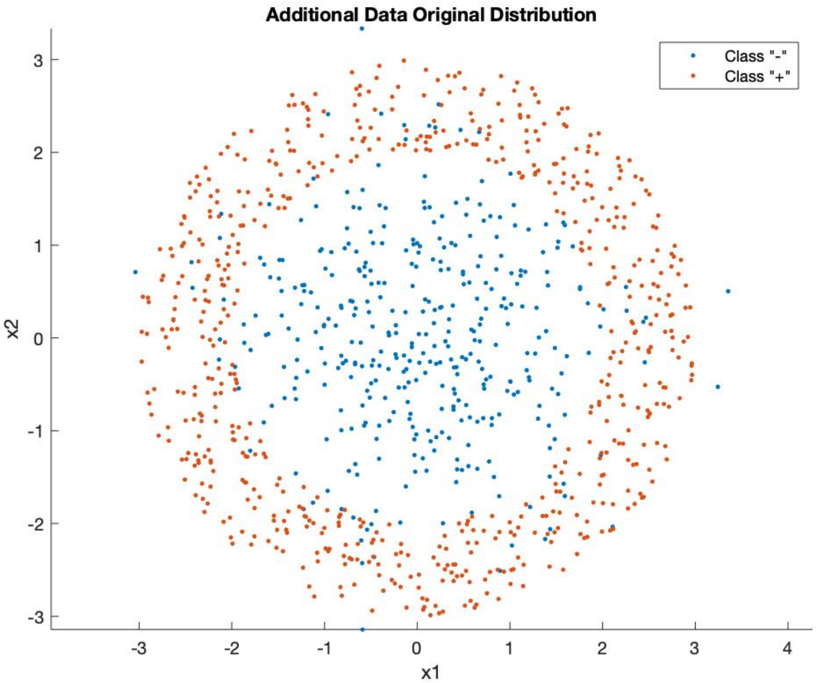


Figure 10: Test Data Original

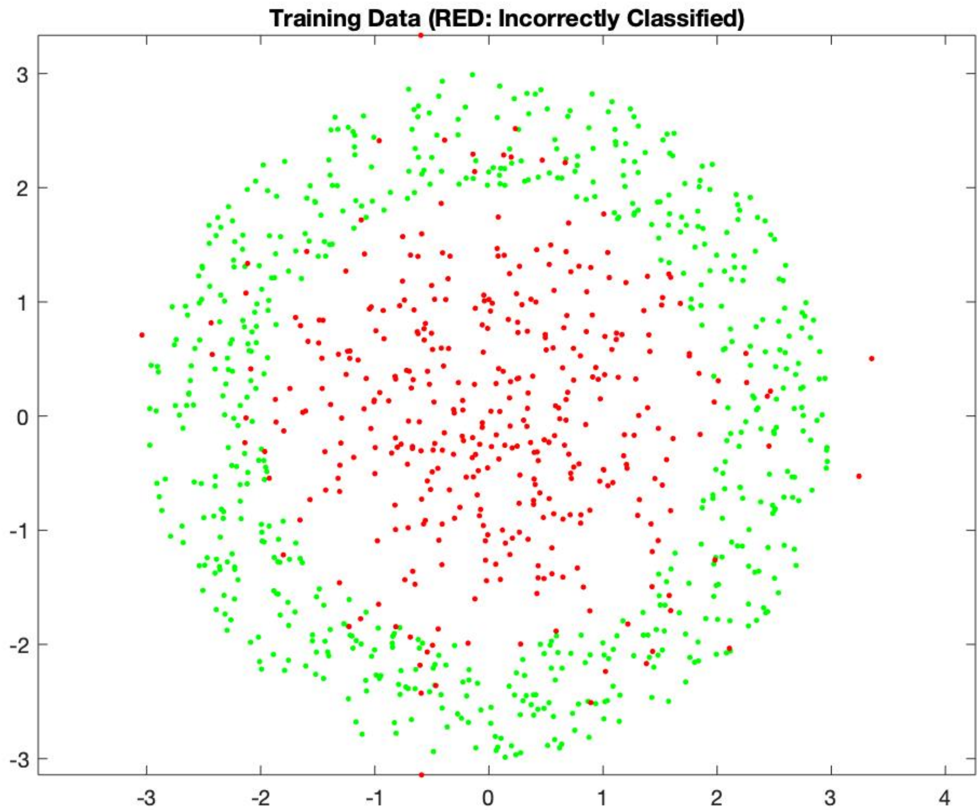


Figure 11: Test Data with Linear SVM

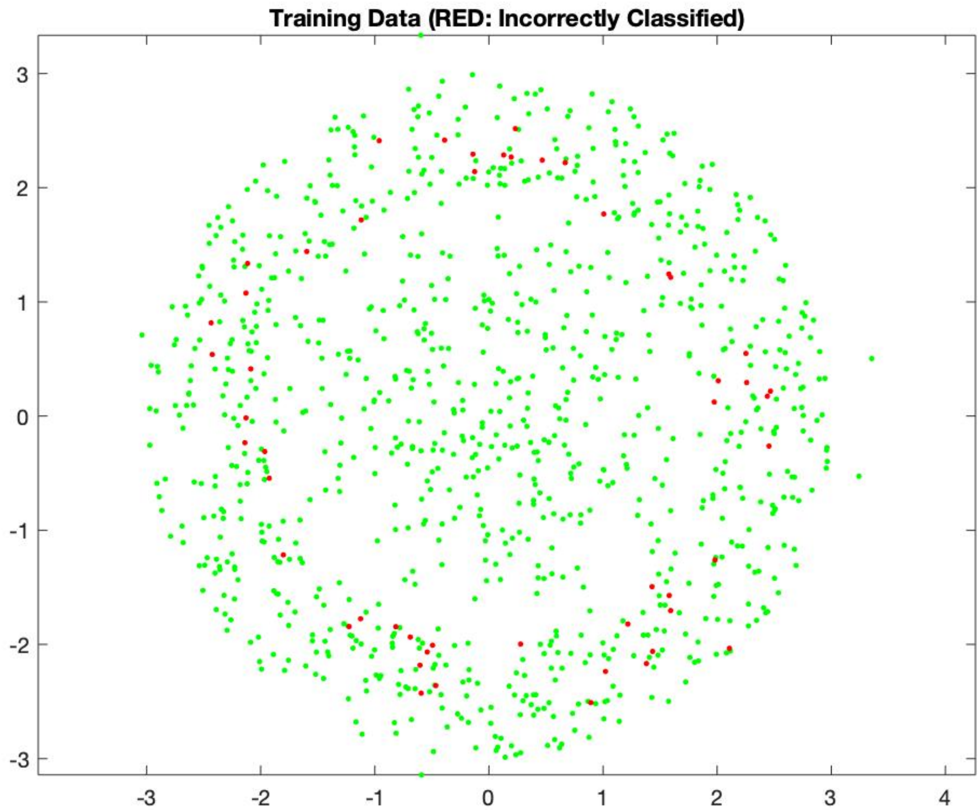


Figure 12: Test data with Gaussian SVM

Our test data results in the following errors:
Linear Test Error: 37.30%
Gaussian Test Error: 5.1%
Once again these errors for the linear match what we would expect based on the priors.

Appendix

1. Problem 1

```
%clear all; close all;
% Import Image
I1 = imread('colorBird.jpg');
I2 = imread('colorPlane.jpg');
% Get width and height of image
Iwidth = length(I1(1,:,1));
Iheight = length(I1(:,1,1));

% Cast image to double for scaling
I1 = cast(I1, 'double');
Ivec = [reshape(I1(:,:,1),[],1),reshape(I1(:,:,2),[],1),reshape(I1(:,:,3),[],1)]; % Reshape image into an nx3 matrix of the form [R G B]
Ivec = Ivec./256; % Divide by max of uint8 (256) to normalize vector
indices = find(Ivec(:,1) == Ivec(:,1));
% [Vertical distance from corner, Horizontal distance from corner]
% Vertical and horizontal distance distance starts at 1
locFeat = [ceil(indices./(Iwidth)), indices-Iwidth*(ceil(indices./(Iwidth))-1)];
% Normalize distances
locFeat1 = [locFeat(:,1)./max(locFeat(:,1)), locFeat(:,2)./max(locFeat(:,2))];

% Develop Feauture Vecotrs in X as follows [Vertical Distance, Horizontal
% Distance, R, G, B]
X = [locFeat1, Ivec];

colors(1,:) = [256 0 0];
colors(2,:) = [0 256 0];
colors(3,:) = [0 0 256];
colors(4,:) = [256 256 0];
colors(5,:) = [0 256 256];
segment = zeros(Iheight,Iwidth,3);
figure(1);
for k = 2:5
    [idx, C] = kmeans(X,k,'MaxIter',10000,'Replicates',5);
    idxReshape = reshape(idx,[],Iwidth);
    segment = zeros(Iheight,Iwidth,3);
    for j = 1:k
        tmp = idxReshape==j;
        tmp = repmat(tmp,[1 1 3]);
```

```
        segment(:,:,1) = segment(:,:,1) + colors(j,1).*tmp(:,:,1);
        segment(:,:,2) = segment(:,:,2) + colors(j,2).*tmp(:,:,2);
        segment(:,:,3) = segment(:,:,3) + colors(j,3).*tmp(:,:,3);
    end
    subplot(2,2,k-1);
    segment = cast(segment,'uint8');
    image(segment);
    xlabel('X Pixels'), ylabel('Y Pixels'), title(strcat('k-means with k = ', num2str(k)));
end

figure(2);
for k = 2:5
    [~, EMpriors, EMmu, EMsigma] = EMforGMM_Edited(length(X(:,1)),length(X(1,:)),X',k);
    idx = evalTopGMM(X, EMpriors, EMmu, EMsigma);
    idxReshape = reshape(idx,[],Iwidth);
    segment = zeros(Iheight,Iwidth,3);
    for j = 1:k
        tmp = idxReshape==j;
        tmp = repmat(tmp,[1 1 3]);
        segment(:,:,1) = segment(:,:,1) + colors(j,1).*tmp(:,:,1);
        segment(:,:,2) = segment(:,:,2) + colors(j,2).*tmp(:,:,2);
        segment(:,:,3) = segment(:,:,3) + colors(j,3).*tmp(:,:,3);
    end
    subplot(2,2,k-1);
    segment = cast(segment,'uint8');
    image(segment);
    xlabel('X Pixels'), ylabel('Y Pixels'), title(strcat('k-means with k = ', num2str(k)));
end


%Begin Plane here
Iwidth = length(I2(1,:,1));
Iheight = length(I2(:,1,1));

% Cast image to double for scaling
I2 = cast(I2, 'double');
Ivec = [reshape(I2(:,:,1),[],1),reshape(I2(:,:,2),[],1),reshape
```

```
    (I2(:,:,3),[],1)]; % Reshape image into an nx3 matrix of the form [R G B]
Ivec = Ivec./256; % Divide by max of uint8 (256) to normalize vector
indices = find(Ivec(:,1) == Ivec(:,1));
% [Vertical distance from corner, Horizontal distance from corner]
% Vertical and horizontal distance distance starts at 1
locFeat = [ceil(indices./(Iwidth)), indices-Iwidth*(ceil(indices./(Iwidth))-1)];
% Normalize distances
locFeat1 = [locFeat(:,1)./max(locFeat(:,1)), locFeat(:,2)./max(locFeat(:,2))];

% Develop Feauture Vecotrs in X as follows [Vertical Distance, Horizontal
% Distance, R, G, B]
X = [locFeat1, Ivec];
segment = zeros(Iheight,Iwidth,3);
figure(3);
for k = 2:5
    [idx, C] = kmeans(X,k,'MaxIter',10000);
    idxReshape = reshape(idx,[],Iwidth);
    segment = zeros(Iheight,Iwidth,3);
    for j = 1:k
        tmp = idxReshape==j;
        tmp = repmat(tmp,[1 1 3]);
        segment(:,:,1) = segment(:,:,1) + colors(j,1).*tmp(:,:,1);
        segment(:,:,2) = segment(:,:,2) + colors(j,2).*tmp(:,:,2);
        segment(:,:,3) = segment(:,:,3) + colors(j,3).*tmp(:,:,3);
    end
    subplot(2,2,k-1);
    segment = cast(segment,'uint8');
    image(segment);
    xlabel('X Pixels'), ylabel('Y Pixels'), title(strcat('k-means with k = ', num2str(k)));
end

figure(4);
for k = 2:5
    [~, EMpriors, EMmu, EMsigma] = EMforGMM_Edited(length(X(:,1)),length(X(1,:)),X',k);
    idx = evalTopGMM(X, EMpriors, EMmu, EMsigma);
```

```
    idxReshape = reshape(idx,[],Iwidth);
    segment = zeros(Iheight,Iwidth,3);
    for j = 1:k
        tmp = idxReshape==j;
        tmp = repmat(tmp,[1 1 3]);
        segment(:,:,1) = segment(:,:,1) + colors(j,1).*tmp(:,:,1);
        segment(:,:,2) = segment(:,:,2) + colors(j,2).*tmp(:,:,2);
        segment(:,:,3) = segment(:,:,3) + colors(j,3).*tmp(:,:,3);
    end
    subplot(2,2,k-1);
    segment = cast(segment,'uint8');
    image(segment);
    xlabel('X Pixels'), ylabel('Y Pixels'), title(strcat('k-means with k = ', num2str(k)));
end


function model = evalTopGMM(x, alpha,mu,sigma)
tGMM = zeros(size(x,1),length(alpha));
for m = 1:length(alpha)
    tGMM(:,m) = alpha(m)*mvnpdf(x,mu(:,m)',sigma(:,:,m));
end
[row, col] = find(tGMM == max(tGMM, [], 2));
model(row) = col;
model = model';
% for m = 1:length(alpha)
%     tGMM(:,m
% end
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
```

2. Problem 2 EM

```
function [logLikelihood, alpha,mu,Sigma] = EMforGMM_Edited(N,d, 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 = d; % Dimension of data coming in.
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 column 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

```
% 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

close all, clear all,
N=1000; n = 2; K=10;
m = 2; % size of feature vector
% Class -1 setup
mu = zeros(1,m)';
Sigma = eye(m);

% Class +1 setup
m = zeros(m,m);
m(:,1) = [2,3]';
m(:,2) = [-pi, pi]';

classPriors = [0.35 0.65];
classPriors1 = [0.35 0.651];
assert(max(cumsum(classPriors)) == 1, 'Priors do not equal 1');
thr = [0,cumsum(classPriors1)];

%Generate Data:
% Defined above N = 1000; % Number of samples to generate for
    each set(10,100,1000,10000)
u = rand(1,N); L = zeros(1,N); x = zeros(2,N);
figure(1),clf, colorList = 'rbgy', hold on;
```

```
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));
    if l == 1
        x(:,indices) = mvnrnd(mu(:,l),Sigma(:,:,l),length(indices))';
        plot(x(1,indices),x(2,indices),'.','MarkerFaceColor',colorList(l)); axis equal;
    end
    if l == 2
        x(:,indices) = [(m(1,1) + (m(2,1)-m(1,1)).*rand(1,length(indices)))', (m(1,2)+ (m(2,2)-m(1,2))*rand(1,length(indices)))']';%r = a + (b-a).*rand(N,1)
        x(:,indices) = [(x(1,indices).*cos(x(2,indices)))',(x(1,indices).*sin(x(2,indices)))']';
        plot(x(1,indices),x(2,indices),'.','MarkerFaceColor',colorList(l)), axis equal;
        %plot(x(1,indices).*cos(x(2,indices)),x(1,indices).*sin(x(2,indices)),'.','MarkerFaceColor',colorList(l)); axis equal;
    end

%     axis([-10 10 -10 10]);
end
xlabel('x1'), ylabel('x2'), legend('Class "-"','Class "+"'), title('Data Original Distribution')

l = 2*(L-1.5);


% Train a Linear 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(-3,7,11);
for CCounter = 1:length(CList)
    [CCounter,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','linear');
        dValidate = SVMk.predict(xValidate')'; % Labels of
            validation data using the trained SVM
        indCORRECT = find(lValidate.*dValidate == 1);
        Ncorrect(k)=length(indCORRECT);
    end
    PCorrect(CCounter)= sum(Ncorrect)/N;
end
disp(strcat('Minimum error linear CV',num2str(min(Ncorrect))));
figure(2), subplot(1,2,1),
plot(log10(CList),PCorrect,'.',log10(CList),PCorrect,'-'),
xlabel('log_{10} C'),ylabel('K-fold Validation Accuracy
    Estimate'),
title('Linear-SVM Cross-Val Accuracy Estimate'), %axis equal,
[dummy,indi] = max(PCorrect(:)); [indBestC, indBestSigma] =
    ind2sub(size(PCorrect),indi);
CBest= CList(indBestC);
SVMBest = fitcsvm(x',l','BoxConstraint',CBest,'KernelFunction',
    'linear');
d = SVMBest.predict(x')'; % Labels of training data using the
    trained SVM
indINCORRECT = find(l.*d == -1); % Find training samples that
    are incorrectly classified by the trained SVM
indCORRECT = find(l.*d == 1); % Find training samples that are
    correctly classified by the trained SVM
figure(2), subplot(1,2,2),
plot(x(1,indCORRECT),x(2,indCORRECT),'g.'), hold on,
plot(x(1,indINCORRECT),x(2,indINCORRECT),'r.'), axis equal,
title('Training Data (RED: Incorrectly Classified)'),
disp('Cross-Fold Validation Gaussian Error');
pTrainingError = length(indINCORRECT)/N, % 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(2), subplot(1,2,2), contour(xGrid,yGrid,zGrid,0); xlabel('x1'), ylabel('x2'), axis equal,
CtrueLinear = CList(indBestC);

% 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(-1,9,23); sigmaList = 10.^linspace(-2,3,23);
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','gaussian','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
disp(strcat('Minimum error Gaussian CV', num2str(min(Ncorrect))));

figure(3), subplot(1,2,1),
contour(log10(CList),log10(sigmaList),PCorrect',50); 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','gaussian','KernelScale',sigmaBest);
d = SVMBest.predict(x')'; % Labels of training data using the trained SVM
indINCORRECT = find(l.*d == -1); % Find training samples that are incorrectly classified by the trained SVM
indCORRECT = find(l.*d == 1); % Find training samples that are correctly classified by the trained SVM
figure(3), subplot(1,2,2),
plot(x(1,indCORRECT),x(2,indCORRECT),'g.'), hold on,
plot(x(1,indINCORRECT),x(2,indINCORRECT),'r.'), axis equal,
title('Training Data (RED: Incorrectly Classified)'),
disp('Cross-Fold Validation Gaussian Error');
pTrainingError = length(indINCORRECT)/N, % Empirical estimate of training error probability
Nx = 10000; Ny = 9900; 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(3), subplot(1,2,2), contour(xGrid,yGrid,zGrid,0); xlabel('x1'), ylabel('x2'), axis equal,
CTrue_Gaussian = CList(indBestC);
SigmaTrue_Guassian = sigmaList(indBestSigma);



%Generate new Data:
% Defined above N = 1000; % Number of samples to generate for each set(10,100,1000,10000)
u = rand(1,N); L = zeros(1,N); x = zeros(2,N);
figure(4),clf, colorList = 'rbgy', hold on;
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));
    if l == 1
        x(:,indices) = mvnrnd(mu(:,l),Sigma(:,:,l),length(
            indices))';
        plot(x(1,indices),x(2,indices),'.','MarkerFaceColor',
            colorList(l)); axis equal;
    end
    if l == 2
        x(:,indices) = [(m(1,1) + (m(2,1)-m(1,1)).*rand(1,
            length(indices)))', (m(1,2)+ (m(2,2)-m(1,2))*rand(1,
            length(indices)))']';%r = a + (b-a).*rand(N,1)
        x(:,indices) = [(x(1,indices).*cos(x(2,indices)))',(x
            (1,indices).*sin(x(2,indices)))']';
        plot(x(1,indices),x(2,indices),'.','MarkerFaceColor',
            colorList(l)), axis equal;
        %plot(x(1,indices).*cos(x(2,indices)),x(1,indices).*sin
            (x(2,indices)),'.','MarkerFaceColor',colorList(l));
            axis equal;
    end

%      axis([-10 10 -10 10]);
end
xlabel('x1'), ylabel('x2'), legend('Class "-"','Class "+"'),
    title('Additional Data Original Distribution')

l = 2*(L-1.5);

SVMk = fitcsvm(xTrain',lTrain,'BoxConstraint',CtrueLinear,'
    KernelFunction','linear');
testValidate = SVMk.predict(x')'; % Labels of validation data
    using the trained SVM
indINCORRECT = find(l.*testValidate == -1); % Find training
    samples that are incorrectly classified by the trained SVM
indCORRECT = find(l.*testValidate == 1);
Ncorrect(k)=length(indCORRECT);
figure(5);
plot(x(1,indCORRECT),x(2,indCORRECT),'g.'), hold on,
plot(x(1,indINCORRECT),x(2,indINCORRECT),'r.'), axis equal,
title('Training Data (RED: Incorrectly Classified)'),
disp('Linear SVM Error New Data');
pTrainingError = length(indINCORRECT)/N, % Empirical estimate
    of training error probability


SVMk = fitcsvm(x',l,'BoxConstraint',CTrue_Gaussian,'
```

```
    KernelFunction','gaussian','KernelScale',SigmaTrue_Guassian)
    ;
dValidate = SVMk.predict(x')'; % Labels of validation data
    using the trained SVM
indINCORRECT = find(l.*dValidate == -1); % Find training
    samples that are incorrectly classified by the trained SVM
indCORRECT = find(l.*dValidate == 1);
Ncorrect(k)=length(indCORRECT);
figure(6);
plot(x(1,indCORRECT),x(2,indCORRECT),'g.'), hold on,
plot(x(1,indINCORRECT),x(2,indINCORRECT),'r.'), axis equal,
title('Training Data (RED: Incorrectly Classified)'),
disp('Gaussian SVM Error New Data');
pTrainingError = length(indINCORRECT)/N, % Empirical estimate
    of training error probability
```