Problem 1

a) $h(x) = w^T x + b = 0$

We need to minimize $\|x - 0\|^2$ which will give the point on the plane closest to the origin.

$$L(x, \lambda) = \|x\|^2 - \lambda(w^T x + b) = 0$$

$$\nabla_x L = 2x - \lambda w^T = 0$$

$$\Rightarrow x = \frac{\lambda w}{2}$$

$\nabla^2_{xx} L = 2$ which is greater than 0 so there is a minimum.

$$\nabla_\lambda L = -w^T x - b = 0$$

$$w^T x + b = 0$$

$$w^T\left(\frac{\lambda w}{2}\right) + b = 0$$

$$\lambda = -\frac{2b}{w^T w}$$

$x_0 = \frac{-bw}{w^T w}$ is the point on the plane that is closest to the origin.

b) when $\|w\| = 1 \Rightarrow w^T w = 1$

That means that $x_0 = \frac{-bw}{1} = -bw$

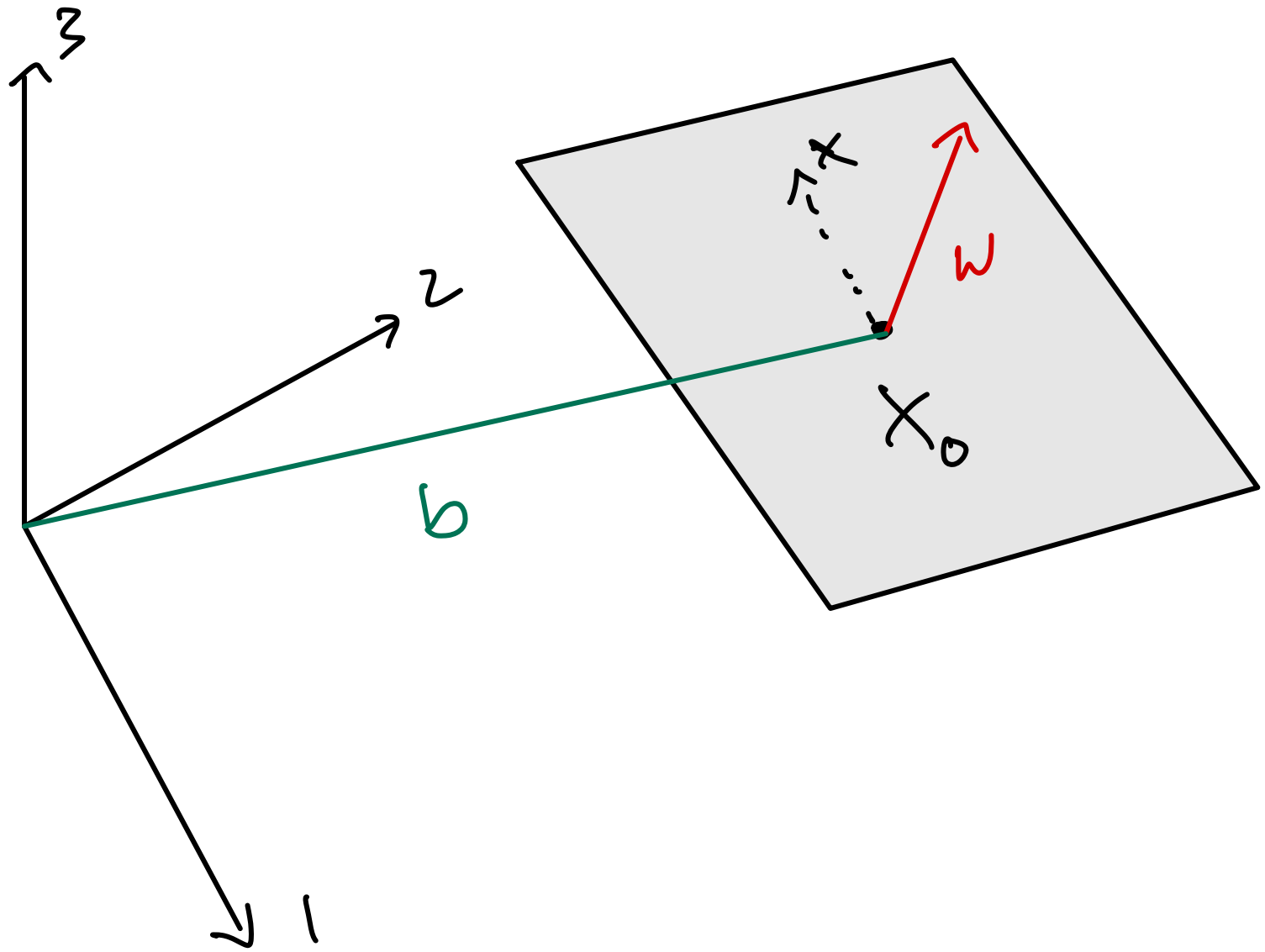
If x, and x_0 are points on the hyperplane, then w must be normal to any points on that hyperplane.

$$w^T(x - x_0) = 0$$

this is valid because

$$w^T(x - x_0) = 0 \Rightarrow w^T x + b = 0$$

$x - x_0$ defines the points on the hyperplane where w is normal to. And b is the distance from that hyperplane if $\|w\| = 1$.


3
2
1
b
x
w
x_0

Problem 2

$$h(x) = -\int p(x) \log[p(x)]\, dx$$

$$\max \; h(x) \quad \text{subject to } E[x] = \mu$$
$$\text{and } \mathrm{Var}[x] = \sigma^2$$

$$\max_{p^*(x)} \Big[h(x) - \lambda\big(E[x] - \mu\big) - \beta\big(\mathrm{Var}[x] - \sigma^2\big) \Big]$$

$$E[x] = \int x\, p(x)\, dx = \mu$$

$$Var[x] = \int (x-\mu)^2\, p(x)\, dx = \sigma^2$$

$$\int p(x)\, dx = 1$$

$$L(p(x), \lambda, \beta)$$

$$h(x) - \lambda(E[x] - \mu) - \beta(Var[x] - \sigma^2)$$

$$\frac{\partial h(x)}{\partial p(x)} = -\log(p(x)) - 1$$

$$\frac{\partial\left[-\lambda\left(E[x]-\mu\right)\right]}{\partial p(x)} = -\lambda(x-0)$$

$$\frac{\partial\left[-\beta\left(Var[x]-\sigma^2\right)\right]}{\partial p(x)} = -\beta\left((x-\mu)^2-0\right)$$

① $\frac{\partial L}{\partial p(x)} = -\log(p(x)) - 1 - \lambda x - \beta(x-\mu)^2 = 0$

$\frac{\partial^2 L}{\partial^2 p(x)} = -\frac{1}{p(x)} < 0$ achieves maximum

$\frac{\partial L}{\partial \lambda} \overset{②}{=} E[x] - \mu = 0$

$\frac{\partial L}{\partial \beta} \overset{③}{=} Var[x] - \sigma^2 = 0$

Using ①:

$$-\log(p(x)) - 1 - \lambda x - \beta(x-\mu)^2 = 0$$

$$\log(p(x)) = -(1 + \lambda x + \beta(x-\mu)^2)$$

$$p^*(x) = \exp\left(-(1 + \lambda x + \beta(x-\mu)^2)\right)$$

b) in order to find the values of λ and β, we need to find the values that would maximize $p^*(x)$. $\lambda = \beta = 0$ in order for $p^*(x)$ to be the highest value.

Problem 3

a) PLA of X

We calculate Σ_x by:

$$\Sigma_x = \frac{1}{n} \sum_{i=1}^{n} (x_i - \mu_x)(x_i - \mu_x)^T$$

$$= \frac{1}{n} X_c X_c^T$$

Then we can use SVD (Σ_x) to find principal components of Σ_x.

If we consider the change of variables $z = x - \mu_x$.

$$\Sigma_z = \frac{1}{n} \sum_{i=1}^{n} (z_i - \mu_z)(z_i - \mu_z)^T$$

$$\mu_z = E[z] = E[x - \mu_x]$$

$$= E[x] - \mu_x = \mu_x - \mu_x = 0$$

$$\Sigma_z = \frac{1}{n} \sum_{i=1}^{n} (z_i - 0)(z_i - 0)^T$$

$$= \frac{1}{n} Z Z^T = \frac{1}{n} (X - \mu_X)(X - \mu_X)^T$$

$$= \frac{1}{n} X_c X_c^T$$

$$\Sigma_X = \Sigma_Z = \frac{1}{n} X_c X_c^T$$

Therefore change of variables does not affect PCA.

b) If $\rho_{ij} \to 1$

$$\Rightarrow 1 = \frac{E[x_i x_j]}{\sqrt{E[x_i^2] E[x_j^2]}}$$

$$E[x_i x_j] \approx \sqrt{E[x_i^2] E[x_j^2]}$$

In PCA, we calculate Σ in which the ith row and jth column is calculated by $\Sigma_{ij} = E[x_i x_j]$ since X has zero mean and

X_i and X_j are highly correlated

$E[x_i^2] \approx E[x_j^2]$, meaning

$E[x_i x_j]$ are large and roughly equal for all i, j.

Since Σ is large for every entry therefore variance is high in every dimension, the largest principal component would be the vector $\mathbf{1}$.

c) $E[z_i] \quad \forall i > 1$

$$= E[\phi_i^T x] = \phi_i^T E[x] = 0$$

We know $E[x] = 0$

$$\Rightarrow \quad E[z_i] = 0 \quad \forall i > 1$$

ϕ_i^T is a constant vector so we can take it out of expectation.

Problem 4

a) $$P_X(x) = P_Y(i=1)\, p(x \mid i=1) + P_Y(i=2)\, p(x \mid i=2)$$

$$= \frac{1}{2} G(x, \mu_1, \Sigma_1) + \frac{1}{2} G(x, \mu_2, \Sigma_2)$$

$$E[x] = E[x \mid y=1] P_Y(1) + E[x \mid y=2] P_Y(2)$$

$$= \mu_1 \cdot \frac{1}{2} + \mu_2 \cdot \frac{1}{2}$$

$$= \frac{1}{2}(\mu_1 + \mu_2)$$

$$\mathrm{Var}[x] = E\left[(x-\mu_x)(x-\mu_x)^T\right]$$

$$= E\left[xx^T - 2x\mu_x + \mu_x\mu_x^T\right]$$

$$= E\left[xx^T\right] - 2\mu_x E[x] + \mu_x\mu_x^T$$

$$= E\left[xx^T\right] - 2\mu_x\mu_x^T + \mu_x\mu_x^T$$

$$= E[XX^T] - \mu_x \mu_x^T$$

$E[XX^T]$

$$= E[XX^T | Y=1] P_Y(1) + E[XX^T | Y=2] P_Y(2)$$

$$= \frac{1}{2}(\Sigma_1 + \mu_1 \mu_1^T) + \frac{1}{2}(\Sigma_2 + \mu_2 \mu_2^T)$$

$$\mu_x \mu_x^T = \frac{1}{2}(\mu_1 + \mu_2) \cdot \frac{1}{2}(\mu_1 + \mu_2)$$

$$= \frac{1}{4}(\mu_1 + \mu_2)(\mu_1 + \mu_2)^T$$

$$\Rightarrow \Sigma_x = \frac{1}{2}(\Sigma_1 + \mu_1 \mu_1^T) + \frac{1}{2}(\Sigma_2 + \mu_2 \mu_2^T)$$

$$= \frac{1}{2}\Sigma_1 + \frac{1}{2}\Sigma_2 + \frac{1}{2}\mu_1 \mu_1^T + \frac{1}{2}\mu_2 \mu_2^T$$

$$- \frac{1}{4}(\mu_1 + \mu_2)(\mu_1 + \mu_2)^T$$

$$= \frac{1}{2}(\Sigma_1 + \Sigma_2) + \frac{1}{4}(\mu_1 - \mu_2)(\mu_1 - \mu_2)^T$$

Problem 4f

$$\mu_x = E[X] = \frac{1}{2}[\mu_1 + \mu_2]$$

$$= \frac{1}{2}\left[\begin{bmatrix}\alpha\\0\end{bmatrix} - \begin{bmatrix}\alpha\\0\end{bmatrix}\right] = \begin{bmatrix}0\\0\end{bmatrix}$$

$$\Sigma_x = \frac{1}{2}[\Sigma_1 + \Sigma_2]$$

$$+ \frac{1}{4}(\mu_1 - \mu_2)(\mu_1 - \mu_2)^T$$

$$= \frac{1}{2}\begin{bmatrix} 2 & 0 \\ 0 & 2\sigma^2 \end{bmatrix}$$

$$+ \frac{1}{4}\left(\begin{bmatrix} 2\alpha \\ 0 \end{bmatrix} \cdot \begin{bmatrix} 2\alpha \\ 0 \end{bmatrix}^T \right)$$

$$= \begin{bmatrix} 1 & 0 \\ 0 & \sigma^2 \end{bmatrix} + \frac{1}{4}\begin{bmatrix} 4\alpha^2 & 0 \\ 0 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 1+\alpha^2 & 0 \\ 0 & \sigma^2 \end{bmatrix}$$

Because Σ_x is diagonal, eigenvectors are the standard basis vectors, and eigenvalues are $1+\alpha^2$ and σ^2.

Therefore $\phi = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ if $1+\alpha^2 > \sigma^2$

or $\phi = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ if $\sigma^2 > 1+\alpha^2$.

LDA

$$w = (\Sigma_1 + \Sigma_2)^{-1}(\mu_1 - \mu_2)$$

$$(2\Gamma)^{-1}\left(\begin{bmatrix}\alpha\\0\end{bmatrix}-\begin{bmatrix}-\alpha\\0\end{bmatrix}\right)$$

$$\begin{bmatrix}\frac{1}{2} & 0\\ 0 & \frac{1}{2\sigma^2}\end{bmatrix}\begin{bmatrix}2\alpha\\0\end{bmatrix}=\begin{bmatrix}\alpha\\0\end{bmatrix}$$

We can see that the linear discriminant is $\begin{bmatrix}\alpha\\0\end{bmatrix}$.

PCA is good for when you want to find the highest variance in your data. If you have labeled data, PCA can throw away this discriminant data, in which, LDA is better.
LDA also assumes that the data is normally distributed. In the case where $\mu = 10$, $\sigma^2 = 2$, the PCA and LDA are the same in this 2D

example. It was clear that there was more variance in the x direction or $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$. In the matlab plot, the PCA and LDA lined up.

Table of Contents

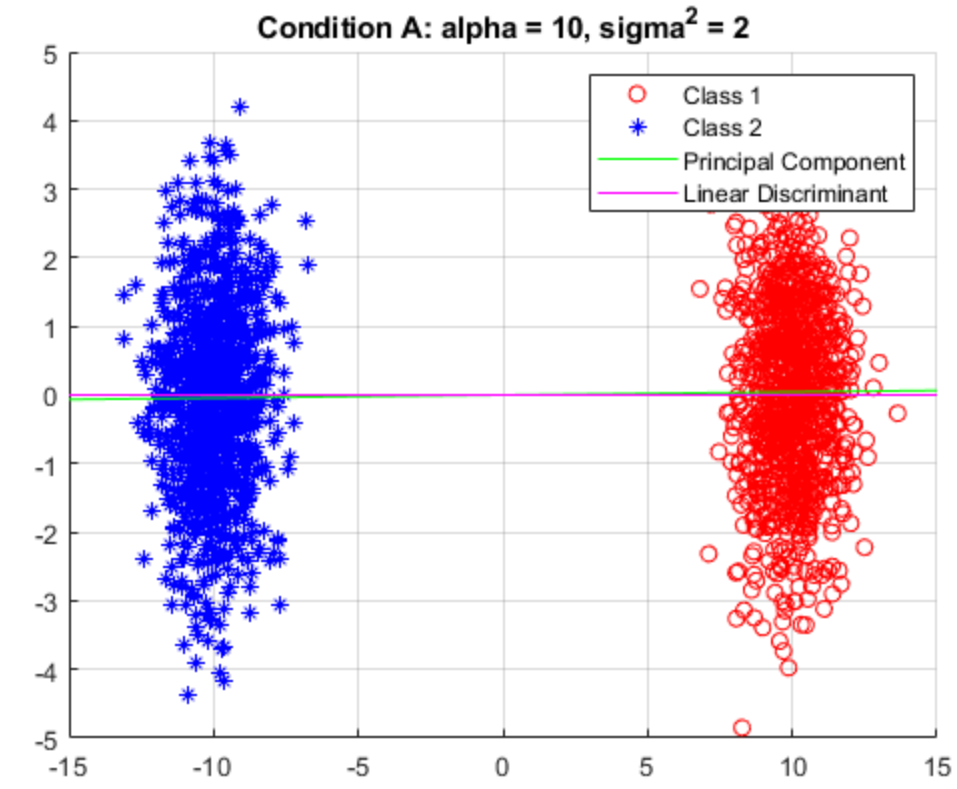

```
%Andy Nguyen
%1/18/2024
%ECE 271B
```

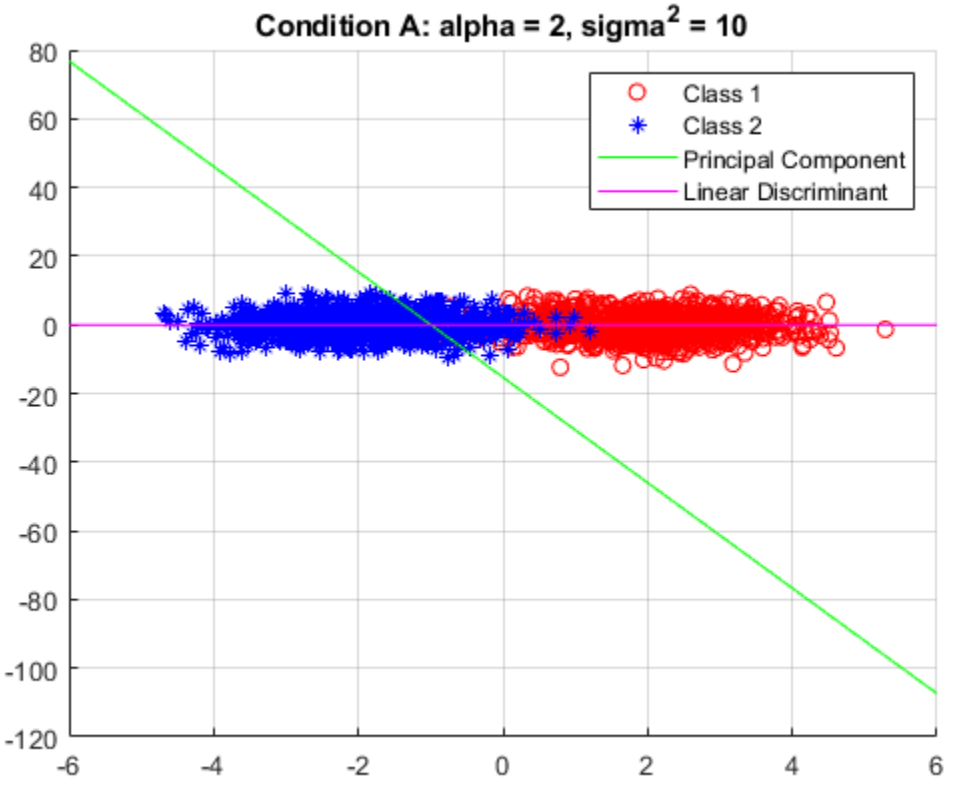
Setup

```
close all; clear; clc;
```

Part 4b, c, and d:

```
[M1, E1, N1, w1] = part4bcd(10, 2, 'Condition A: alpha = 10, sigma^2 = 2');
[M2, E2, N2, w2] = part4bcd(2, 10, 'Condition A: alpha = 2, sigma^2 = 10');
```


Condition A: alpha = 10, sigma^2 = 2
Class 1
Class 2
Principal Component
Linear Discriminant
5
4
3
2
1
0
-1
-2
-3
-4
-5
-15
-10
-5
0
5
10
15


Condition A: alpha = 2, sigma^2 = 10
Class 1
Class 2
Principal Component
Linear Discriminant
80
60
40
20
0
-20
-40
-60
-80
-100
-120
-6
-4
-2
0
2
4
6

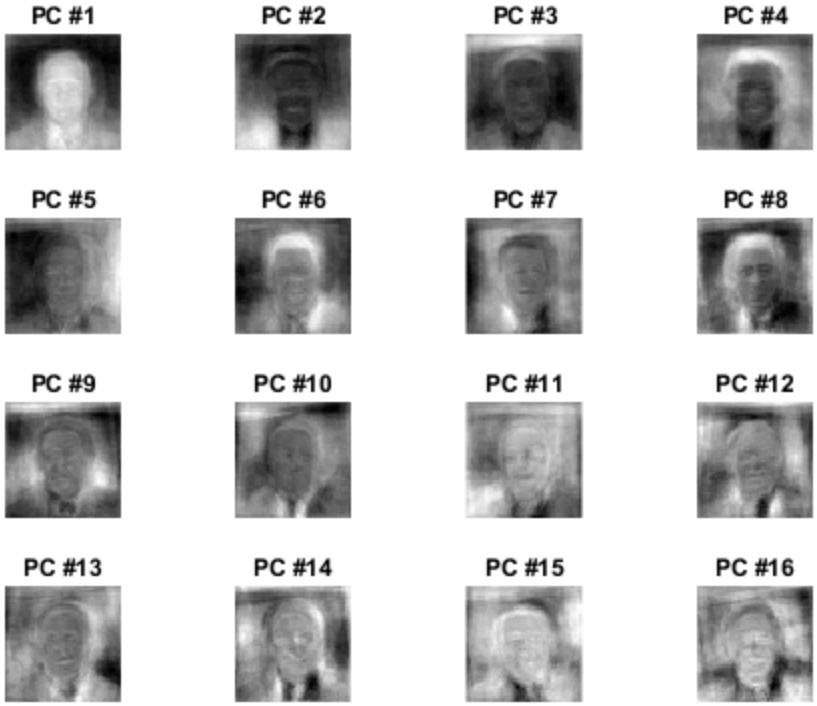
Part 4e:

```
% approach to dimensionality reduction is not always a good one. It is
% possible for the PCA to throw away all the discrimnant info. Although we
% are keeping the dimnesions of largest variance, we might run the risk of
% throwing away important information stored in the other dimensions. In
% terms of classification, LDA seems to perform better because it projects
% the data in a way that maximizes the distance between classes. In the
% example plots, we can see that the PCA and LDA lines up with the sigma
% being less than the means (condition A). This shows that PCA and LDA
% performance are similar here. However, in condition B, when the sigma was
% greatly larger than the means, we can see that LDA was better are showing
% the dimension that better separated the data.
```

Part 5a: compute PCA of the data and make a 4x4 plot showing 16 principal components of largest variance

Get training set

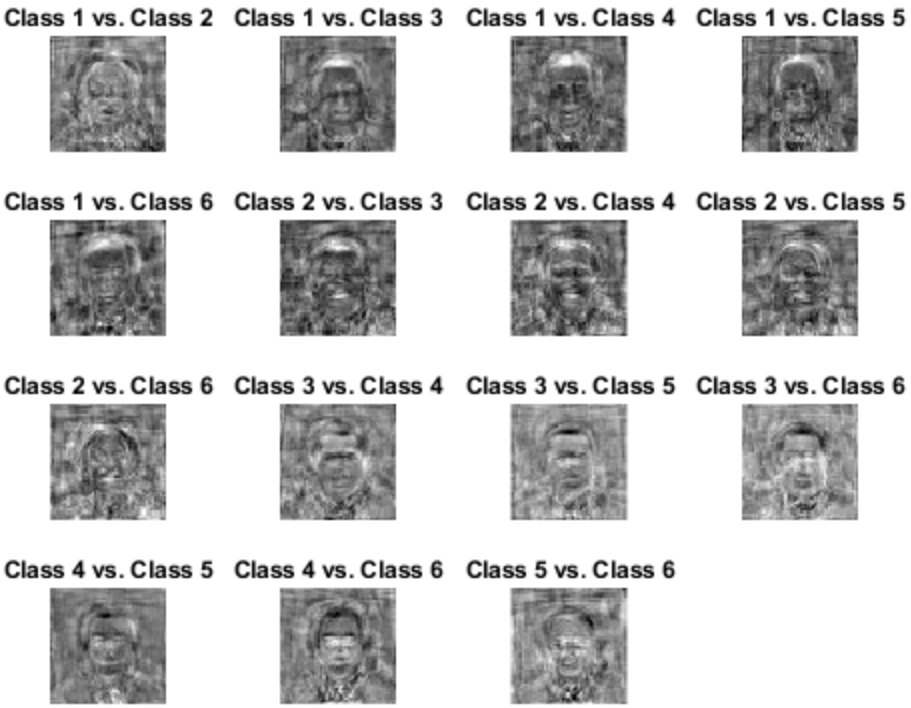
```
mainDir = 'X:\ECE\ECE271B\hw1\trainset';
[folderList, ~] = extractImages(mainDir);
numImages = sum(cellfun(@(c) length(c), folderList));
%reshape 50x50 matrix into 2500 dimensional vector and add to final matrix
X = zeros(2500, numImages);
imageIndex = 1;
for j = 1:length(folderList)
    for i = 1:length(folderList{j})
        imgVector = reshape(folderList{j}{i}, [2500, 1]);
        X(:, imageIndex) = imgVector;
        imageIndex = imageIndex + 1;
    end
end
%make values of A from 0 to 1
X = X/255.0;
X_mean = sum(X, 2)/size(X,2);
X_centered = X-X_mean;
[M, E, N] = svd(X_centered');

top16PC = diag(E(1:16, 1:16));
%create 4x4 plot
figure();
for i = 1:16
    subplot(4,4,i)
    eigenface = mat2gray(reshape(N(:,i), [50,50]));
    imshow(eigenface)
    title(sprintf('PC #%i', i))
end
```



Part 5b

```
figure();
plotIndex = 1;
phi30 = N(:, 1:30); %PCA of 30 principal components (2500 x 30)
for class1Index = 1:length(folderList)-1
    for class2Index = class1Index+1:length(folderList)
        A = zeros(2500, length(folderList{class1Index}));
        B = zeros(2500, length(folderList{class2Index}));

        for i = 1:length(folderList{class1Index})
            imgVectorA = reshape(folderList{class1Index}{i}, [2500, 1]);
            A(:, i) = imgVectorA;
            imgVectorB = reshape(folderList{class2Index}{i}, [2500, 1]);
            B(:, i) = imgVectorB;
        end
        A = A/255.0; % 2500 x 40
        B = B/255.0;
        A_mean = sum(A, 2) / size(A, 2);
        B_mean = sum(B, 2) / size(B, 2);
        A_cov = A*A';
        B_cov = B*B';
        %RDA where gamma =1
        gamma = 1;
        S_w = A_cov + B_cov + gamma*eye(size(A_cov));
```

```
%         S_b = (A_mean - B_mean) * (A_mean - B_mean)';
        w(:, plotIndex) = inv(S_w)*(A_mean - B_mean);
        subplot(4,4, plotIndex)
        eigenface_b = mat2gray(reshape(w(:, plotIndex), [50, 50]));
        imshow(eigenface_b)
        hold on;

        title(sprintf("Class %i vs. Class %i", class1Index, class2Index))
%         legend(sprintf("Class %i", class1Index), sprintf("Class %i",
 class2Index))


        %Part 5e code:
        A_PCA = phi30'*A; %30 dimensions x 40 features
        B_PCA = phi30'*B; %30 x 40
        A_PCA_mean = sum(A_PCA, 2) / size(A_PCA,2); % 30 x 1
        B_PCA_mean = sum(B_PCA, 2) / size(B_PCA,2);
        A_PCA_cov = A_PCA*A_PCA'; %30 x 30
        B_PCA_cov = B_PCA*B_PCA';
        S_w_PCA = A_PCA_cov + B_PCA_cov; %30 x 30
        w_PCA(:, plotIndex) = inv(S_w_PCA)*(A_PCA_mean-B_PCA_mean); %30 x 15
 (when loop is done)

        plotIndex = plotIndex + 1;
    end

end
```



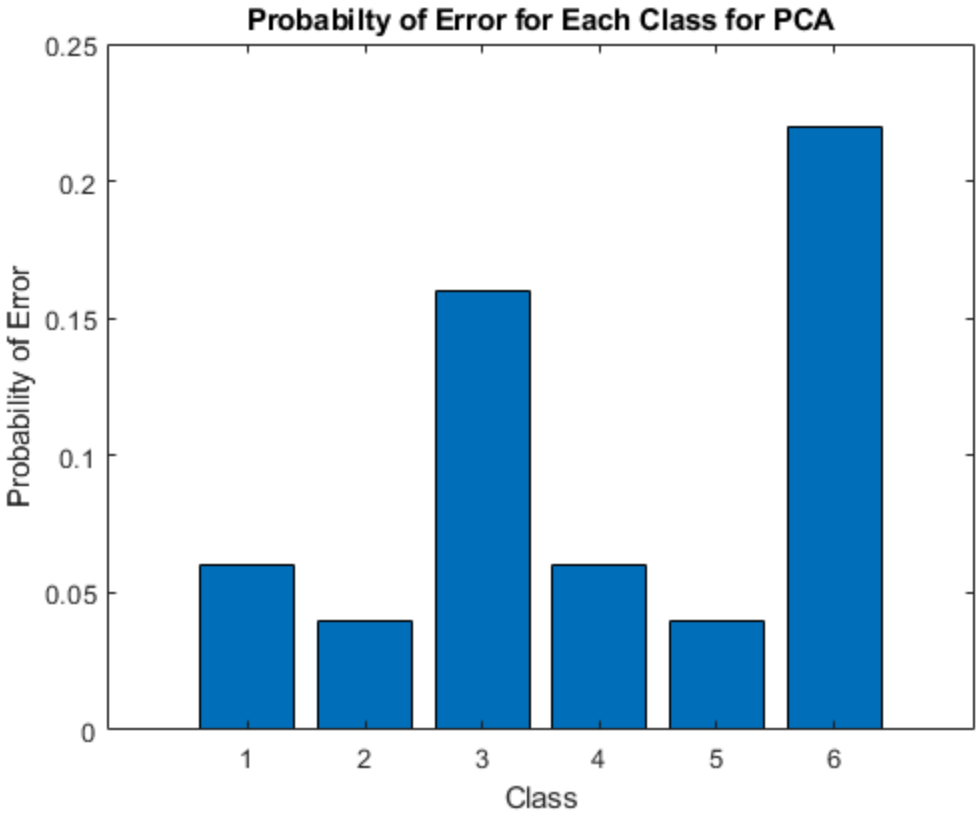
Part C

```
phi = N(:, 1:16);

for class1Index = 1:length(folderList)
    X = zeros(2500, length(folderList{class1Index}));
    for i = 1:length(folderList{class1Index})
        imgVectorX = reshape(folderList{class1Index}{i}, [2500, 1]);
        X(:, i) = imgVectorX;
    end
    X = double(X) / 255.0;
    z_PCA = phi'*X;
    z_LDA = w'*X;
    %for PCA
    mu_classPCA{class1Index} = mean(z_PCA,2);
    z0_PCA = z_PCA - mu_classPCA{class1Index};
    sigma_classPCA{class1Index} = (z0_PCA*z0_PCA') / (size(z_PCA,2)-1);

    %for LDA
    mu_classLDA{class1Index} = mean(z_LDA, 2);
    z0_LDA = z_LDA - mu_classLDA{class1Index};
    sigma_classLDA{class1Index} = (z0_LDA*z0_LDA') / (size(z_LDA,2)-1);

    %for PCA + LDA (k = 30)
    z_PCA_LDA = w_PCA'*phi30'*X;
```

```
    mu_class_PCA_LDA{class1Index} = mean(z_PCA_LDA, 2);
    z0_PCA_LDA = z_PCA_LDA - mu_class_PCA_LDA{class1Index};
    sigma_class_PCA_LDA{class1Index} = (z0_PCA_LDA*z0_PCA_LDA') /
 (size(z0_PCA_LDA, 2)-1);

end

%iterate through test folder list
testDir = 'X:\ECE\ECE271B\hw1\testset';
[testFolderList, testImgNames] = extractImages(testDir);

[averageProbErrorPCA, probErrorPCA] =
 calc_prob_error_gauss_classifier(mu_classPCA, sigma_classPCA, phi,
 testFolderList, testImgNames);
title('Probabilty of Error for Each Class for PCA')
averageProbErrorPCA
```

averageProbErrorPCA =

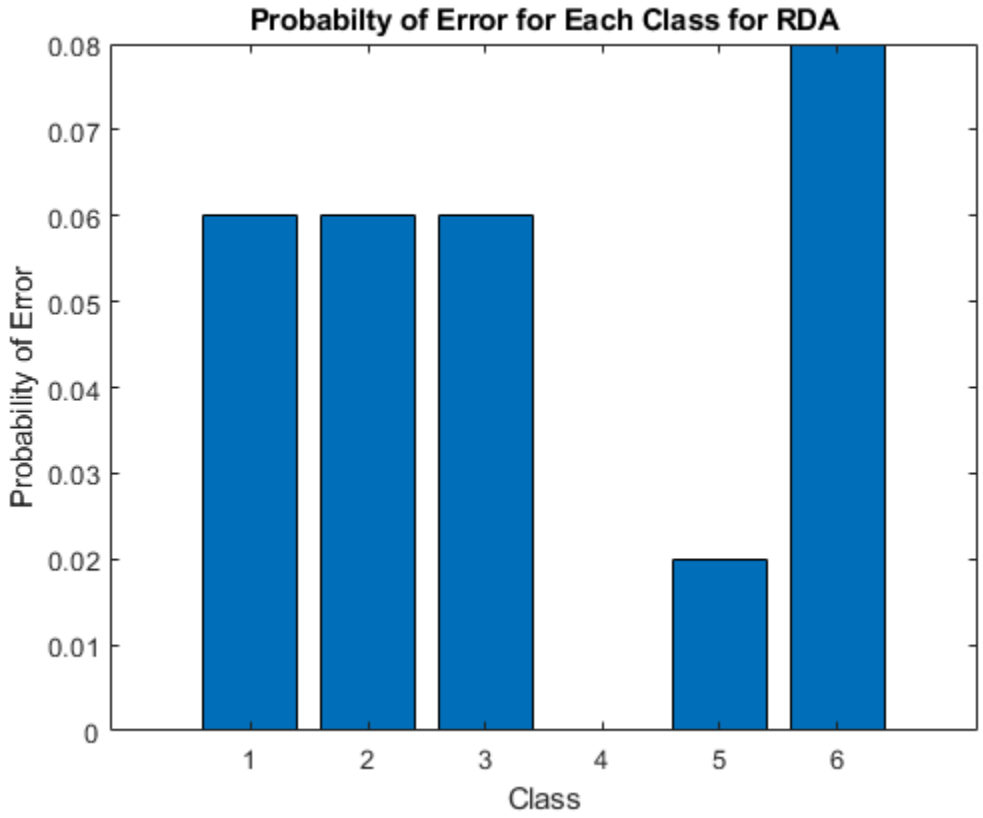
0.0967



Part 5d

```
[averageProbErrorRDA, probErrorLDA] =
 calc_prob_error_gauss_classifier(mu_classLDA, sigma_classLDA, w,
 testFolderList, testImgNames);
title('Probabilty of Error for Each Class for RDA')
averageProbErrorRDA
```

```
averageProbErrorRDA =

    0.0467
```



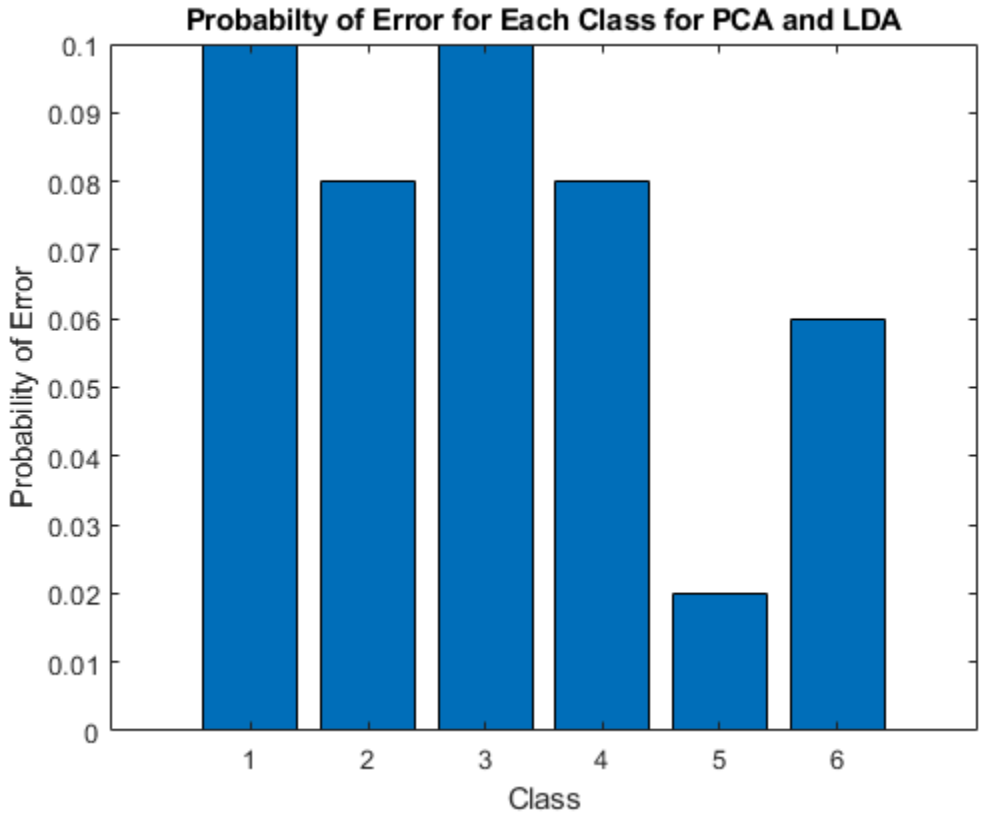
Part 5e

From these three graphs, we can see that just doing PCA performed the worst. RDA seems to have performed the best and PCA + LDA seems to be second. We can see that although PCA can help clean up the data and reduce dimensionality, LDA can be more beneficial in terms of this gaussian classifier example. RDA is good when we need to address singular scatter matricies. PCA + LDA is also a good alternative to address singular covariances. In other examples, PCA might be more beneficial as a technique, however, in the gaussian classifier case, it seems that RDA outperforms the other techniques.

```
[averageProbError_PCA_and_LDA, probError_PCA_and_LDA] =
 calc_prob_error_gauss_classifier(mu_class_PCA_LDA, sigma_class_PCA_LDA,
 phi30*w_PCA, testFolderList, testImgNames);
```

```
title('Probabilty of Error for Each Class for PCA and LDA')
averageProbError_PCA_and_LDA
```

```
averageProbError_PCA_and_LDA =

    0.0733
```



functions

```
function [] = plotLine(pointA, pointB, color)
%plots infinite line from pointA to pointB

xlim = get(gca, 'XLim');
m = (pointB(2)-pointA(2))/(pointB(1)-pointA(1));
n = pointB(2)*m - pointA(2);
y1 = m*xlim(1) + n;
y2 = m*xlim(2) + n;
line([xlim(1) xlim(2)],[y1 y2], 'Color', color)

end

function [M, E, N, w] = part4bcd(alpha, sigma, titleString)

%part 4b
```

```
mu1 = [alpha; 0];
mu2 = -mu1;
Gamma_cov = [1, 0; 0, sigma];
numSamples = 1000;
samplesClass1 = mvnrnd(mu1, Gamma_cov, numSamples);
samplesClass2 = mvnrnd(mu2, Gamma_cov, numSamples);
figure();
hold on;
grid on;
title(titleString)
plot(samplesClass1(:,1), samplesClass1(:,2), 'ro')
plot(samplesClass2(:,1), samplesClass2(:,2), 'b*')
%part 4c
X = [samplesClass1', samplesClass2'];
mean_X = sum(X,2)/size(X,2);
Xc = X - mean_X;
[M, E, N] = svd(Xc');
plotLine([0,0], [N(1,1), N(1,2)], 'g')

%part 4d

w = inv(Gamma_cov + Gamma_cov)*(mu2 - mu1);
plotLine([0,0], w, 'm')
legend('Class 1', 'Class 2', 'Principal Component', 'Linear Discriminant')

end

function min_i = gauss_classify(phi, imageVector, class_means, class_sigmas)
min_i = -1; %not valid
min_prob = inf;
for index = 1:length(class_means)
    prob = (phi'*imageVector -
 class_means{index})'*pinv(class_sigmas{index})*(phi'*imageVector -
 class_means{index})+log(det(class_sigmas{index}));
    if (prob < min_prob)
        min_prob = prob;
        min_i = index;
    end
end
end

function [averageProbError, probError] =
 calc_prob_error_gauss_classifier(mu_class, sigma_class, phi, testFolderList,
 testImgNames)
totalClassImg = zeros(6,1);
correctClassCount = zeros(6,1);
for n = 1:length(testFolderList)
    for m = 1:length(testFolderList{n})
        imgVector = double(reshape(testFolderList{n}{m}, [2500, 1])) / 255.0;
        testImgNames{n}{m};
        label = str2double(testImgNames{n}{m}(8));
        min_i = gauss_classify(phi, imgVector, mu_class, sigma_class);
        totalClassImg(label) = totalClassImg(label) + 1;
        if (label == min_i)
```

```
            correctClassCount(label) = correctClassCount(label) + 1;
        end
    end
end

probError = zeros(6,1);
figure()
for i  = 1:length(totalClassImg)
    probError(i) = 1-correctClassCount(i) / totalClassImg(i);
end
bar([1:6], probError);
title('Probabilty of Error for Each Class')
ylabel('Probability of Error')
xlabel('Class')
averageProbError = 1-sum(correctClassCount) / sum(totalClassImg);
end
```

Published with MATLAB® R2022b