1 Problem 1

1.1 Part A: ERM classification

1.1.1 Decision Rule

The minimum expected risk classification rule can be expressed as:

$$\frac{p(x|L=1)}{p(x|L=0)} \underset{D2}{\overset{D1}{\gtrless}} \frac{P(L=0)(\lambda_{10}-\lambda_{00})}{P(L=1)(\lambda_{01}-\lambda_{11})} \tag{1}$$

Substituting class priors $P(L=0)=0.7$ and $P(L=1)=0.3$ reduces to (2)

$$\frac{p(x|L=1)}{p(x|L=0)} \underset{D2}{\overset{D1}{\gtrless}} \frac{2.33(\lambda_{10}-\lambda_{00})}{(\lambda_{01}-\lambda_{11})} \tag{2}$$

1.1.2 ROC Curve for ERM Classifier with True Knowledge of Data

Fig.1 shows the ROC curve for the ERM classifier. The Optimal empirically selected Tau $\tau_{opt}^{emp}=1.234$ for a random run of 10K samples is marked on the figure with a green star. It's corresponding $p(error;\tau_{opt})=0.0305$. Note that $\tau_{opt}^{emp}=\log\gamma_{opt}^{emp}$. Fig. 2 shows $p(error;\tau)$ with τ selected from midpoints of discriminant scores.

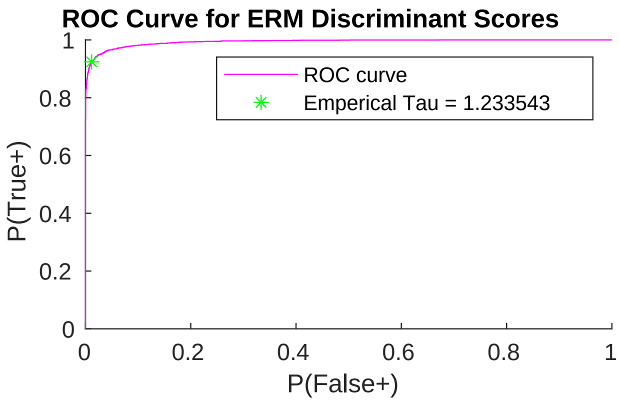

Figure 1: ERM ROC

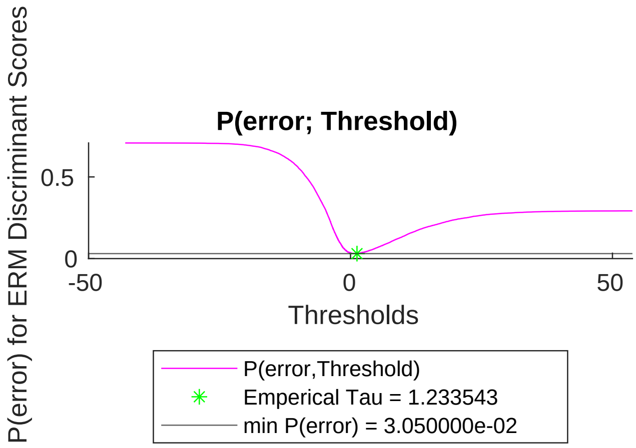

Figure 2: P(error; τ) vs τ (naive Bayesian)

1.1.3 Optimal Theoretical Gamma

The min probability of error occurs at the optimal theoretical τ_{opt}^{theo} which can be calculated by assuming 0-1 loss for the cost matrix. This is expressed as each entry in the cost matrix being equal to a Kronecker delta indexed by decision d and class label l. Applying (3) to decision rule (2) results in $\gamma_{opt}^{theo}=2.33$ with $\tau_{opt}^{theo}=log(\gamma_{opt}^{theo})=0.846$. The achieved error performance of both τ_{opt}^{emp} and τ_{opt}^{emp} is given in Table 4. The difference between them is negligible and their error performance is similar.

$$\lambda_{dl}=1-\delta_{dl} \tag{3}$$

	τ_{opt}	$p(error;\tau_{opt})$
Theoretical	0.846	0.0314
Empirical	1.234	0.0305

Table 1: τ_{opt} Theoretical vs Empirical optimal error performance comparison

1.2 Part B: Naive Bayesian Assumption

1.2.1 Modification to covariance matrices

Applying a naive Bayesian assumption to Q1 results in approximating each the conditional probability of each class by assuming the covariance matrices are diagonal. Effectively, the resulting covariance matrices for $C0$ and $C1$ are then expressed by the below matrices.

$$C0 = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix} C1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix} \tag{4}$$

1.2.2 ROC curve and error performance

The naive Bayesian assumption results in a classifier with lower error performance. This is reflected in Table 3 where the empirically estimated threshold beats both the naive Bayesian optimal threshold and the theoretical optimal threshold.

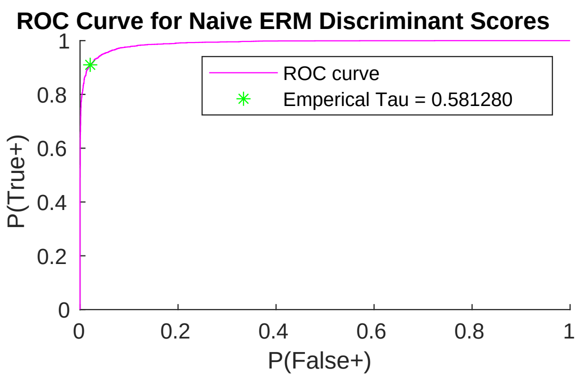

Figure 3: ERM ROC (naive Bayesian)

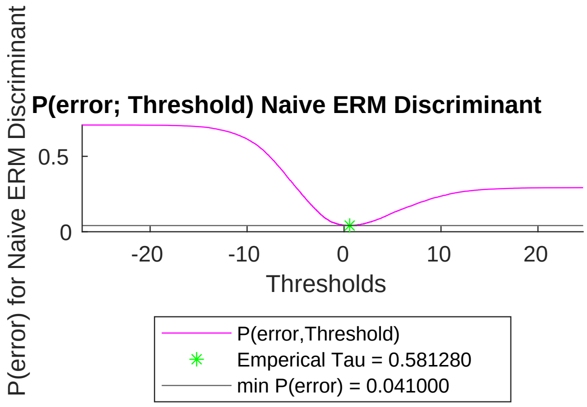

Figure 4: P(error; τ) vs τ (naive Bayesian)

	τ_{opt}	$p(error;\tau_{opt})$
Empirical	1.234	0.0305
Theoretical	0.846	0.0314
Empirical (Naive Bayesian)	0.581	0.041

Table 2: τ_{opt} Theoretical vs Empirical optimal error performance comparison for naive Bayesian assumption

1.3 Part C: Fisher LDA

1.3.1 Data Projection

After calculating the within-class and between-class scatter matrices, the high dimensional data can be reduced to just 1 dimension as illustrated in fig. 5.

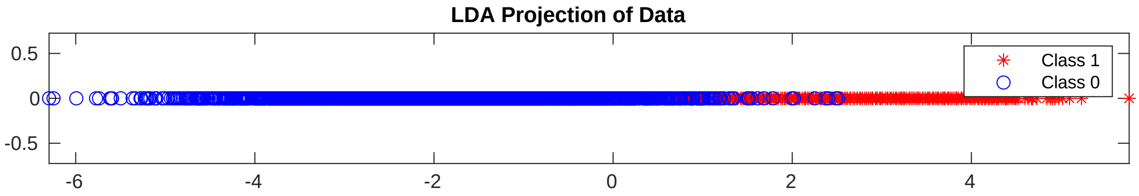

Figure 5: LDA Projection of Data

1.3.2 ROC and error performance

The LDA classifier's error performance was inferior relative to ERM either at the theoretical or empirical optimum, however, for this particular run of 10k samples, it performed better than the naive Bayesian classifier as reflected in table

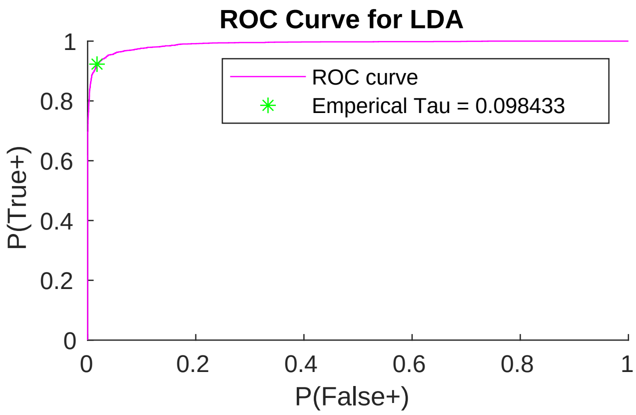

Figure 6: LDA ROC

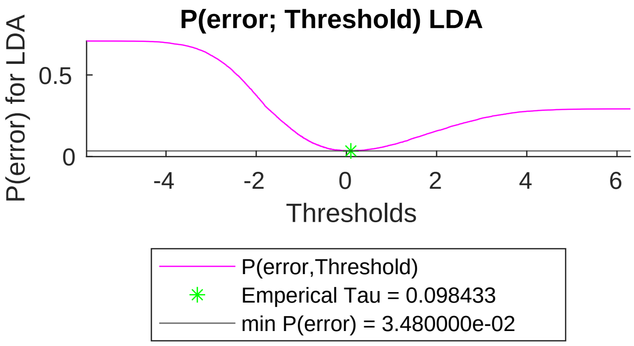

Figure 7: P(error; τ) vs τ (naive Bayesian)

	τ_{opt}	$p(error; \tau_{opt})$
Empirical	1.234	0.0305
Theoretical	0.846	0.0314
Empirical (LDA)	0.098	0.034
Empirical (Naive Bayesian)	0.581	0.041

Table 3: τ_{opt} Theoretical vs Empirical optimal error performance comparison for naive Bayesian assumption

1.4 Extra Work Done For Fun: 2D scatter plot of data with decision boundary

This work was done for fun. If it's wrong please ignore! Given the high dimensionality of the data, visualising both the data and the decision boundary for the ERM classifier is challenging. A scatter plot for all 2D combinations of the n-component of the data results in $\binom{n}{2}$ plots. If the components of the data are independent then all possible combinations don't have to be presented together. However, in Q1's case, the original covariance matrices are not diagonal, so the components are dependent. For the sake of brevity, let's just pick 2 combinations for illustration because the contour plot part is the more interesting. The x coordinates of the decision boundary are defined by the values of x that satisfy $log(P(x|L = 1)) - log(P(x|L = 0)) = log(\gamma_{opt})$. We can then define a new function $\omega(x) = log(P(x|L = 1)) - log(P(x|L = 0)) - \tau_{opt}$ where $\omega(x) = 0$ is the optimal ERM decision boundary. If x was 2D we could superimpose the contour plots of $\omega(x)$ at level=0 to define the optimal decision boundary for each class for the 2 components of x. However, x is 4D. To be able to present the decision boundary we can split the 4D data problem into 6 2D data problems where the optimal decision boundary is evaluated for each 2D combination of the x's components. This results in 6 ERM classifiers with different decision boundaries for each pair of x's components. To represent this new data, the resulting mean $\hat{\mu}$ and covariance matrices $\hat{\Sigma}$ are sub-matrices of the original x mean μ and covariance Σ matrices for each class conditional probability. In the case of any 2 component pairs (X_i, X_j) the mean and covariance matrices $\hat{\mu}_{ij} = \mu_i, \mu_j, \hat{\Sigma} = \Sigma[i : j; i : j]$ (assuming $j > i$). This approach has been applied in the code available in the Appendix and has resulted in the graphs in fig. 8 and fig.9.
I'm operating under the assumption that applying the 6 ERM classifiers will produce the same classification result as 1 monolithic 4D ERM classifier.

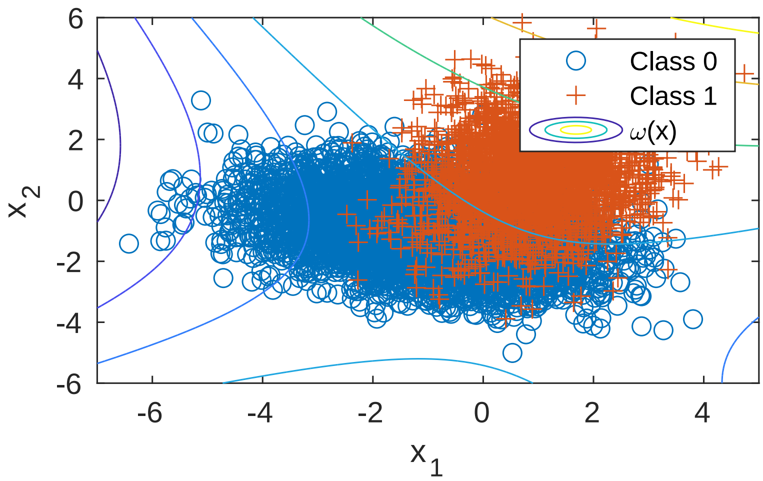

Figure 8: First and Second Components of X with their own decision boundary

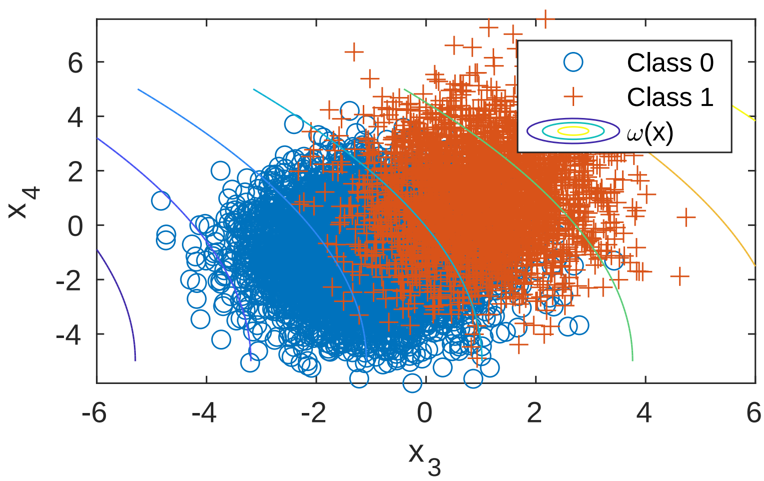

Figure 9: Third and Fourth Components of X with their own decision boundary

2 Problem 2

2.1 Part A

2.1.1 Parameters Chosen

The following parameters were chosen to satisfy the requirements of Q2. The means were chosen to guarantee placement of the distributions at the corners of a square. The diagonals of class conditional covariance matrices where chosen to guarantee distribution symmetry. The Eigen values of the covariance matrices where increased to reduce separation between the distributions. There is a slight asymmetry between the overall size of the class conditional distributions to make the problem slightly more exciting.

$$Class_0 : \Sigma_0 = \lambda_0 I, \lambda_0 = 16.6667, \mu_0 = [0; 10; 10] \tag{5}$$

$$Class_1 : \Sigma_1 = \lambda_1 I, \lambda_1 = 21.6667, \mu_1 = [0; 10; -10] \tag{6}$$

$$Class_2 : \Sigma_2 = \lambda_2 I, \lambda_2 = 26.6667, \mu_2 = [0; -10; -10] \tag{7}$$

$$Class_3 : \Sigma_3 = \lambda_3 I, \lambda_3 = 31.6667, \mu_3 = [0; -10; 10] \tag{8}$$

2.1.2 Data 3D Scatter Plot

A Labeled 3D Scatter plot of the data is provided in fig. 10 as well as 2D projections of the data using PCA in figures 11, 12. The asymmetry in size between the class conditionals can be seen in both fig. 10 and fig. 12 with the purple class having a wider radius.

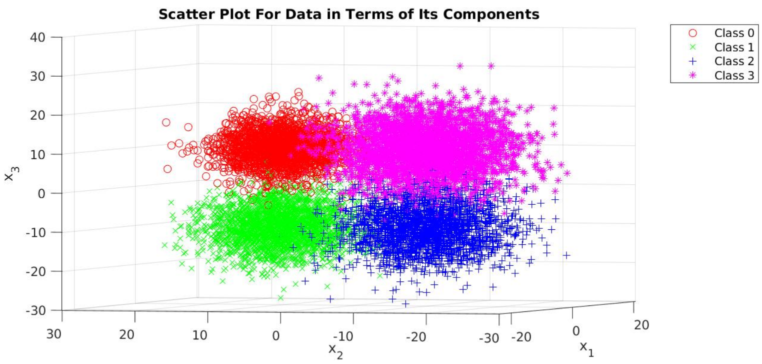

Figure 10: 3D Scatter plot of generated data

2.1.3 Extra Work: Effect of PCA projection on the data

If all components of PCA are used, PCA produces projection of the data that eliminate dependency between individual components which results in the new components having identity covariance matrices. Given that the original data's components were already independent this has no effect on the original data. What does have an effect on the data post PCA is whitening. Whitening scales results in the projected data's diagonal entries being scaled by the square root of their Eigen values which causes the 4 class conditional distributions to have the same radius as seen in fig. 13, 11 and 12. This results in data that's more separable if the whitened data were to be used instead of the original data. This idea has been explored in the code of Q2 of the appendix. The code compares the error performance of an ERM classifier operating on the original data vs an ERM classifier operating on the whitened data. In the interest of space this result was removed from this report (mostly because the assumption that ERM operating on whitened data wasn't always true, but it's still there).

2.1.4 Confusion Matrix and Decision Rule

The decision rule is given in eq. (9) where $lambda_{dl}$ is are loss matrix entries corresponding to different decision label pairs. The confusion matrix is given in fig. 14. Given the asymmetry between the sizes of the class conditionals the probability of making the correct decision is for each class is slightly different. Given that both the separation and the asymmetry between the conditionals is controllable in the code, more interesting results with wider disparities between the probabilities in the confusion matrix can be generated.

$$D(x) = argmin_{d \in \{1,2,3,\ldots,C\}} \sum_{l=1}^{C} \lambda_{dl} P(x|L = l)P(L = l) \tag{9}$$

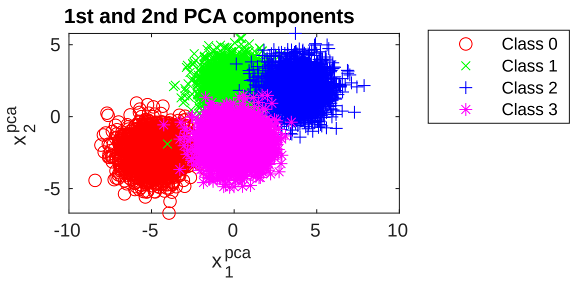

Figure 11: 2D Scatter PCA plot of generated data with first and second components of x

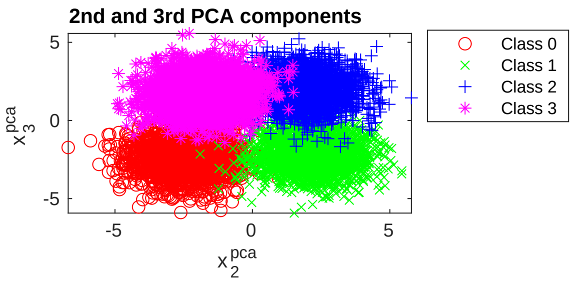

Figure 12: 2D Scatter PCA plot of generated data with second and third components of x

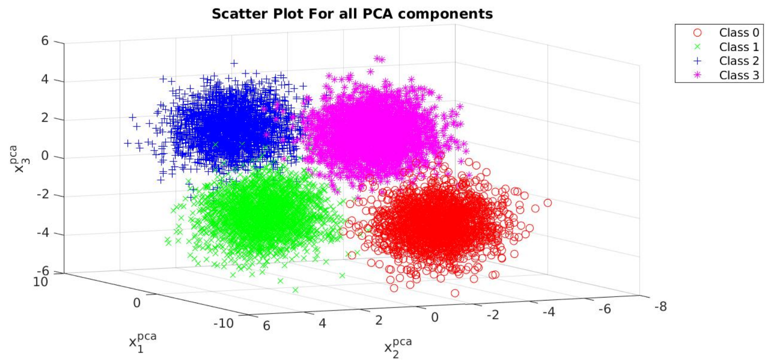

Figure 13: 3D Scatter PCA plot of generated data

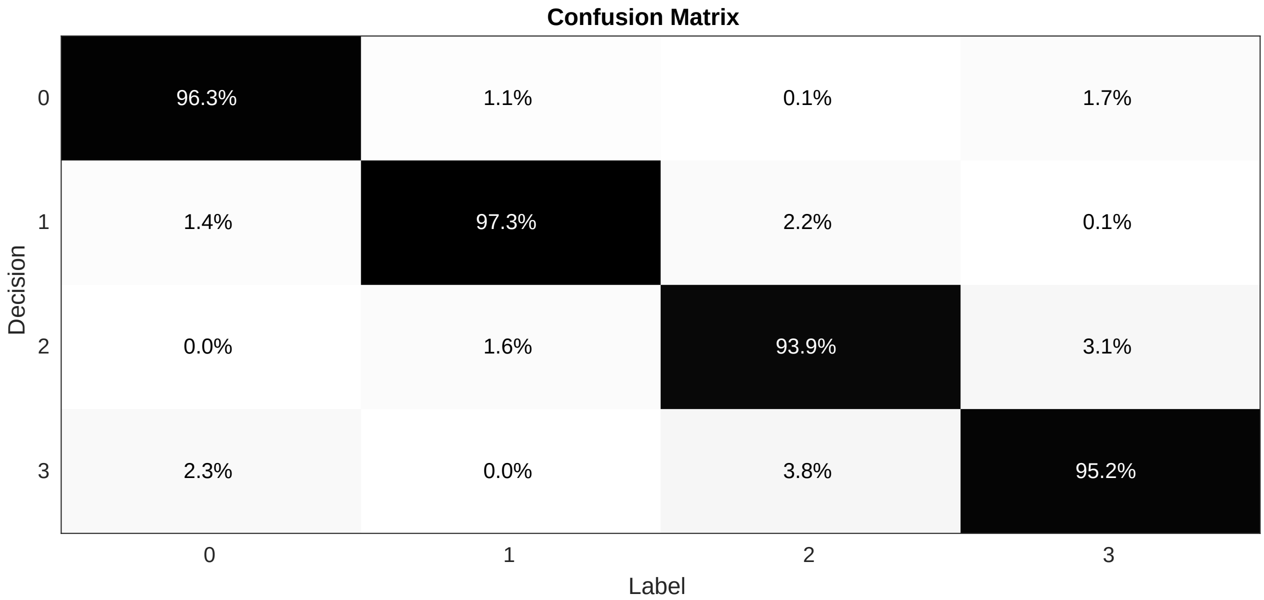

Figure 14: Confusion Matrix

2.1.5 Classification Results Using 2D and 3D PCA projections

fig. 15 and 16 show the effect of applying ERM on resulting decisions with correct classifications being colored green and incorrect ones colored red. Given the difficulty of visually identifying the decision boundary of ERM from the projections, fig. 17 shows all 3 components of PCA post classification. The PCA projections were used here because they negate the asymmetry between the class conditionals and make it easier to identify the decision boundary from the correct/incorrect classifications.

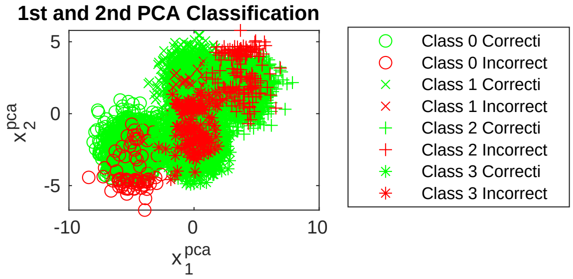

Figure 15: 2D Scatter PCA plot of classified data with first and second components of x

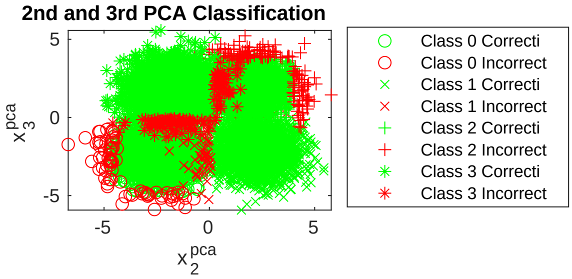

Figure 16: 2D Scatter PCA plot of classified data with second and third components of x

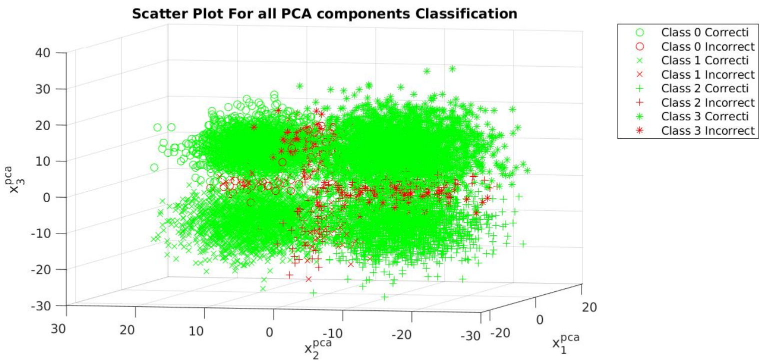

Figure 17: 3D Scatter PCA plot of classified data

2.2 Part B: Estimating minimum expected risk

The estimated minimum expected risk can be calculated from eq. (10). The estimated minimum expected risk for both the 0-1 loss matrix and the cost matrix given in Part B is given in Table

$$E[\min Risk(D = d|L = l)] \approx \frac{1}{N} \sum_{i=1}^{N} \lambda_{dl} \tag{10}$$

Cost Matrix	E[min $Risk(D = d|L = l)$]
0-1 Loss	0.0446
Part-B	0.1606

Table 4: τ_{opt} Theoretical vs Empirical optimal error performance comparison

3 Appending

Code is also available here: Public Github Repo

3.1 Q1.m

```matlab
clear all, close all,

%% ========================= Conditional PDF Paramemter Setup ========================= %%

n = 4; % number of feature dimensions
N = 10000; % number of iid samples
mu(:,1) = [-1;-1;-1;-1];
mu(:,2) = [1;1;1;1];
Sigma(:,:,1) = [2 -0.5 0.3 0; -0.5 1 -0.5 0; 0.3 -0.5 1 0; 0 0 0 2];
Sigma(:,:,2) = [1 0.3 -0.2 0; 0.3 2 0.3 0; -0.2 0.3 1 0; 0 0 0 3];

% mu(:,1) = 5*rand(1,n);
% mu(:,2) = 5*rand(1,n);
% A1 = 3*(rand(n,n)-0.5);
% A2 = 3*(rand(n,n)-0.5);
% Sigma(:,:,1) = A1*A1';
% Sigma(:,:,2) = A2*A2';

p = [0.7,0.3]; % class priors for labels 0 and 1 respectively

labels = rand(1,N) >= p(1);
Nc = [length(find(labels==0)),length(find(labels==1))]; % number of samples from each class
x = zeros(n,N); % save up space

%% ========================= Dataset Generation ========================= %%

for i = 1:N
    if(labels(i)==0)
        x(:,i) = mvnrnd(mu(:,1),Sigma(:,:,1));
    else
        x(:,i) = mvnrnd(mu(:,2),Sigma(:,:,2));
    end
end

%% ========================= ERM descriminator score evaluation ========================= %%

p_x_given_l_equals_0 = evalGaussian(x,mu(:,1),Sigma(:,:,1));
p_x_given_l_equals_1 = evalGaussian(x,mu(:,2),Sigma(:,:,2));

descriminant_score_ERM = log(p_x_given_l_equals_1./p_x_given_l_equals_0);

%% ========================= ROC plot ========================= %%

[PfpERM,PfnERM,PtpERM,PtnERM,PerrorERM,thresholdListERM] = ROCcurve(descriminant_score_ERM,labels);
figure(1), clf,
subplot(5,2,3), hold on, plot(PfpERM,PtpERM,'m'),
xlabel('P(False+)'),ylabel('P(True+)'), title('ROC Curve for ERM Discriminant Scores'),
subplot(5,2,4), hold on, plot(thresholdListERM,PerrorERM,'m'),
xlabel('Thresholds'), ylabel('P(error) for ERM Discriminant Scores'), title('P(error; Threshold)')

%% ========================= Theoretical and Empirical optimal descriminator threshold ========================= %%
lambda = [0 1;1 0]; % loss values
theoretical_gamma = log(((lambda(2,1)-lambda(1,1))*p(1))/((lambda(1,2)-lambda(2,2))*p(2))); % threshold
min_emperical_PerrorERM = min(PerrorERM)
min_perrorERM_index = find(PerrorERM == min(PerrorERM));
optimal_emperical_gamma = thresholdListERM(min_perrorERM_index);
PtpERM_at_optimal_emperical_gamma = PtpERM(min_perrorERM_index);
PfpERM_at_optimal_emperical_gamma = PfpERM(min_perrorERM_index);

tau = theoretical_gamma;
decisions = (descriminant_score_ERM >= tau);
min_PerrorERM_theoretical = sum(decisions~=labels)/length(labels)

%% ========================= Threshold plots on ROC curve and P(error;Tau) curve ========================= %%
subplot(5,2,4), hold on
plot(optimal_emperical_gamma(1),min(PerrorERM),'g*'), hold on,
yline(min(PerrorERM))
legend('P(error,Threshold)', ['Emperical Tau = ' num2str(optimal_emperical_gamma(1),'%02f')], ['min P(error) = ' num2str(min(PerrorERM),'%02d')]),

subplot(5,2,3), hold on
plot(PfpERM_at_optimal_emperical_gamma,PtpERM_at_optimal_emperical_gamma,'g*'), hold on,
legend('ROC curve', ['Emperical Tau = ' num2str(optimal_emperical_gamma(1),'%02f')]),

%% ========================= 2D Scatter Plots for X components ========================= %%
label0_indexes = find(labels==0);
label1_indexes = find(labels==1);

subplot(5,2,1),
```

```
plot(x(1,label0_indexes),x(2,label0_indexes),'o'), hold on,
plot(x(1,label1_indexes),x(2,label1_indexes),'+'),
lgnd = legend('Class 0','Class 1');
title('Data and their true labels For First And Second Component of x'),
xlabel('x_1'), ylabel('x_2'),

subplot(5,2,2),
plot(x(3,label0_indexes),x(4,label0_indexes),'o'), hold on,
plot(x(3,label1_indexes),x(4,label1_indexes),'+'),
lgnd = legend('Class 0','Class 1');
title('Data and their true labels For Third And Fourth Component of '),
xlabel('x_3'), ylabel('x_4');


%% ======================== Contour Plot of Decision Boundry ======================== %%
% to draw 2D contors, let's pretend the problem has been reduced to
% 2 problems with 2 class ERM and x in 2D, mu and sigma are then submatricies of the
% original matricies. The optimal theoretical gamma is independent of
% number of components of x. Therefore if we just look at the covariance
% matrix of only 2 components at a time plus their means, the optimal
% decision boundry will be defined at the same level (0) of the 3D surface
% score = log(g1) - log(g2) - log(gamma). Contour plots this surface can
% then be plotted with a mesh grid from the min and max of the pair of
% components being looked at. This process can be done for all component
% pairs and it should given that that they are not independent from each
% other, however, it has only been done for 2 possible component pairs for
% illustration.

aGrid = linspace(floor(min(x(1,:))),ceil(max(x(1,:))),100);
bGrid = linspace(floor(min(x(2,:))),ceil(max(x(2,:))),100);
cGrid = linspace(floor(min(x(3,:))),ceil(max(x(3,:))),100);
dGrid = linspace(floor(min(x(4,:))),ceil(max(x(4,:))),100);

[a, b] = meshgrid(aGrid,bGrid);
[c, d] = meshgrid(bGrid,cGrid);

discriminantScoreGridValues = log(evalGaussian([a(:)';b(:)'],mu(1:2,2),Sigma(1:2,1:2,2)))-log(evalGaussian([a(:)';b(:)'],mu(1:2,1),Sigma(1:2,1:2,1))) - log(theoretical_gamma);
minDSGV = min(discriminantScoreGridValues);
maxDSGV = max(discriminantScoreGridValues);
discriminantScoreGrid = reshape(discriminantScoreGridValues,100,100);

subplot(5,2,1),
contour(aGrid,bGrid,discriminantScoreGrid,[minDSGV*[0.9,0.6,0.3],0,[0.3,0.6,0.9]*maxDSGV]); % plot equilevel contours of the discriminant function
lgnd = legend('Class 0','Class 1', 'Contours of discriminant function');
lgnd.Location = 'southeast';

discriminantScoreGridValues = log(evalGaussian([c(:)';d(:)'],mu(3:4,2),Sigma(3:4,3:4,2)))-log(evalGaussian([c(:)';d(:)'],mu(3:4,1),Sigma(3:4,3:4,1))) - log(theoretical_gamma);
minDSGV = min(discriminantScoreGridValues);
maxDSGV = max(discriminantScoreGridValues);
discriminantScoreGrid = reshape(discriminantScoreGridValues,100,100);

subplot(5,2,2),
contour(bGrid,cGrid,discriminantScoreGrid,[minDSGV*[0.9,0.6,0.3],0,[0.3,0.6,0.9]*maxDSGV]); % plot equilevel contours of the discriminant function
lgnd = legend('Class 0','Class 1', 'Contours of discriminant function');
lgnd.Location = 'southeast';


%% ======================== Applying Naive Bayesian Assumptions ======================== %%
Sigmahat(:,:,1) = Sigma(:,:,1) .* eye(4);
Sigmahat(:,:,2) = Sigma(:,:,2) .* eye(4);

naive_p_x_given_l_equals_0 = evalGaussian(x,mu(:,1),Sigmahat(:,:,1));
naive_p_x_given_l_equals_1 = evalGaussian(x,mu(:,2),Sigmahat(:,:,2));

naive_descriminant_score_ERM = log(naive_p_x_given_l_equals_1./naive_p_x_given_l_equals_0);

%% ======================== ROC plot for Naive Bayesian ======================== %%
[naive_PfpERM,naive_PfnERM,naive_PtpERM,naive_PtnERM,naive_PerrorERM,naive_thresholdListERM] = ROCcurve(naive_descriminant_score_ERM,labels);
subplot(5,2,5), hold on,
plot(naive_PfpERM,naive_PtpERM,'m'),
xlabel('P(False+)'),ylabel('P(True+)'), title('ROC Curve for Naive ERM Discriminant Scores');

subplot(5,2,6), hold on, plot(naive_thresholdListERM, naive_PerrorERM,'m'),
xlabel('Thresholds'), ylabel('P(error) for Naive ERM Discriminant'), title('P(error; Threshold) Naive ERM Discriminant');

%% ======================== Threshold plots on ROC curve and P(error;Tau) (Naive Bayesian) ======================== %%
naive_min_perrorERM_index = find(naive_PerrorERM == min(naive_PerrorERM));
naive_min_perrorERM = min(naive_PerrorERM)
naive_optimal_emperical_gamma = naive_thresholdListERM(naive_min_perrorERM_index);
naive_PtpERM_at_optimal_emperical_gamma = naive_PtpERM(naive_min_perrorERM_index);
naive_PfpERM_at_optimal_emperical_gamma = naive_PfpERM(naive_min_perrorERM_index);

subplot(5,2,6), hold on
plot(naive_optimal_emperical_gamma(1),min(naive_PerrorERM),'g*'), hold on,
yline(min(naive_PerrorERM))
```

```
legend('P(error,Threshold)', ['Emperical Tau = ' num2str(naive_optimal_emperical_gamma(1),'%02f')], ['min P(error) = ' num2str(min(naive_PerrorERM),'%02f')]),

subplot(5,2,5), hold on
plot(naive_PfpERM_at_optimal_emperical_gamma(1),naive_PtpERM_at_optimal_emperical_gamma(1),'g*'), hold on,
legend('ROC curve', ['Emperical Tau = ' num2str(naive_optimal_emperical_gamma(1),'%02f')]),

%% ========================= LDA sample mean and covariance estimate ========================= %%
muhat(:,1) = mean(x(:,label0_indexes),2);
muhat(:,2) = mean(x(:,label1_indexes),2);
Sigmahat(:,:,1) = cov(x(:,label0_indexes)');
Sigmahat(:,:,2) = cov(x(:,label1_indexes)');

%% ========================= LDA between class and within class scatter matrix calculation ========================= %%
Sb = (muhat(:,1)-muhat(:,2))*(muhat(:,1)-muhat(:,2))'; Sw = Sigmahat(:,:,1) + Sigmahat(:,:,2);

%% ========================= LDA weights calculation ========================= %%
[V,D] = eig(Sw\Sb); [~,ind] = sort(diag(D),'descend');
w = V(:,ind(1)); % Fisher LDA projection vector at greatest eigen value
y1 = w'*x(:,label0_indexes); y2 = w'*x(:,label1_indexes);
if mean(y2)<=mean(y1), w = -w; end % push label0 projections to the left

%% ========================= LDA Projection Plot ========================= %%
subplot(5,2,[7 8]), plot(y1(1,:),zeros(1,size(y1,2)),'r*'); hold on;
plot(y2(1,:),zeros(1,size(y2,2)),'bo'); axis equal,
legend('Class 1','Class 0');
title("LDA Projection of Data");

%% ========================= LDA ROC Plot ========================= %%
y=w'*x;
[LDA_Pfp,LDA_Pfn,LDA_Ptp,LDA_Ptn,LDA_Perror,LDA_thresholdList] = ROCcurve(y,labels);
subplot(5,2,9), hold on,
plot(LDA_Pfp, LDA_Ptp,'m'),
xlabel('P(False+)'),ylabel('P(True+)'), title('ROC Curve for LDA');

subplot(5,2,10), hold on, plot(LDA_thresholdList, LDA_Perror,'m'),
xlabel('Thresholds'), ylabel('P(error) for LDA'), title('P(error; Threshold) LDA');

%% ========================= Threshold plots on LDA ROC curve and LDA P(error;Tau) curve ========================= %%
subplot(5,2,10), hold on
min_LDA_Perror = min(LDA_Perror)
LDA_min_perror_index = find(LDA_Perror == min(LDA_Perror));
LDA_optimal_emperical_gamma = LDA_thresholdList(LDA_min_perror_index);
plot(LDA_optimal_emperical_gamma(1),min(LDA_Perror),'g*'), hold on,
yline(min(LDA_Perror))
legend('P(error,Threshold)', ['Emperical Tau = ' num2str(LDA_optimal_emperical_gamma(1),'%02f')], ['min P(error) = ' num2str(min(LDA_Perror),'%02d')]),

LDA_Ptp_at_optimal_emperical_gamma = LDA_Ptp(LDA_min_perror_index);
LDA_Pfp_at_optimal_emperical_gamma = LDA_Pfp(LDA_min_perror_index);
subplot(5,2,9), hold on
plot(LDA_Pfp_at_optimal_emperical_gamma(1),LDA_Ptp_at_optimal_emperical_gamma(1),'g*'), hold on,
legend('ROC curve', ['Emperical Tau = ' num2str(LDA_optimal_emperical_gamma(1),'%02f')]),

%% ========================= Helper Functions ========================= %%
function g = evalGaussian(x,mu,Sigma)
% Evaluates the Gaussian pdf N(mu,Sigma) at each coumn of X
[n,N] = size(x);
C = ((2*pi)^n * det(Sigma))^(-1/2);
E = -0.5*sum((x-repmat(mu,1,N)).*(inv(Sigma)*(x-repmat(mu,1,N))),1);
g = C*exp(E);
end

function [Pfp,Pfn,Ptp,Ptn,Perror,thresholdList] = ROCcurve(discriminantScores,labels)
[sortedScores,~] = sort(discriminantScores,'ascend');
thresholdList = [min(sortedScores)-eps,(sortedScores(1:end-1)+sortedScores(2:end))/2, max(sortedScores)+eps];
Ptp = zeros(1,length(thresholdList));
Ptn = zeros(1,length(thresholdList));
Pfp = zeros(1,length(thresholdList));
Pfn = zeros(1,length(thresholdList));
Perror = zeros(1,length(thresholdList));
for i = 1:length(thresholdList)
    tau = thresholdList(i);
    decisions = (discriminantScores >= tau);
    Ptp(i) = length(find(decisions==1 & labels==1))/length(find(labels==1));
    Pfp(i) = length(find(decisions==1 & labels==0))/length(find(labels==0));
    Ptn(i) = length(find(decisions==0 & labels==0))/length(find(labels==0));
    Pfn(i) = length(find(decisions==0 & labels==1))/length(find(labels==1));
    Perror(i) = sum(decisions~=labels)/length(labels);
end
end

function [y] = pca(x, sigma, mu)

% If original distribution parameters not passed then
% use sample-based estimates of mean and covariance matrix
if ~exist('sigma', 'var')
    Sigmahat = cov(x');
```

```
else
    Sigmahat = sigma;
end

if ~exist('mu', 'var')
    muhat = mean(x,2)';
else
    muhat = mu';
end

% make data 0 mean
xzm = x - muhat*ones(size(x));

% Get the eigenvectors (in Q) and eigenvalues (in D) of the
% estimated covariance matrix
[Q,D] = eig(Sigmahat);

% Sort the eigenvalues from large to small, reorder eigenvectors
% accordingly as well.
[d,ind] = sort(diag(D),'descend');
Q = Q(:,ind);
D = diag(d);

% Calculate the principal components (in y)
% Also whiten components so their norms are 1
y = D^(-1/2)*Q'*xzm;
end
```

3.2 Q2.m

```
clear all, close all,

%% ========================= Conditional PDF Paramemter Setup ========================= %%
n = 3; % number of feature dimensions
N = 10000; % number of iid samples
L = 4; % number of labels
label_text = {'0', '1', '2', '3'}';
mu(:,1) = [0; 10;10];
mu(:,2) = [0; 10;-10];
mu(:,3) = [0; -10;-10];
mu(:,4) = [0; -10;10];

assymmetry = 5;
seperation = 0.06;
proximity = 1/seperation;
lambda = repmat(proximity,1,L)+(0:L-1).*assymmetry;

Sigma(:,:,1) = lambda(1).*eye(n);
Sigma(:,:,2) = lambda(2).*eye(n);
Sigma(:,:,3) = lambda(3).*eye(n);
Sigma(:,:,4) = lambda(4).*eye(n);

priors = [0.2, 0.25, 0.25, 0.3]; % class priors for labels 0 -> 3
p = cumsum(priors);

%% ========================= Data Generation ========================= %%
[x, labels] = generateData4Gaussians(n, N, p, mu, Sigma);

%% ========================= Data Scatter Plot ========================= %%
figure(1), clf,
subplot(6,5,[1 2 6 7]);
mShapes = 'ox+*.';
mColors = 'rgbmy';
for l = 0:L-1
    scatter3(x(1,labels == l), x(2,labels == l), x(3,labels == l), strcat(mShapes(l+1),mColors(l+1)
        )), hold on;
end
title('Scatter Plot For Data in Terms of Its Components');
xlabel('x_1'), ylabel('x_2'), zlabel('x_3'),
legend('Class 0', 'Class 1', 'Class 2', 'Class 3');

%% ========================= Data 2D Projection with PCA Scatter Plot ========================= %%
for l = 0:L-1
    x_label = x(:,labels==l);
    pca_components = pca(x_label, Sigma(:,:,l+1), mu(:, l+1));
    subplot(6,5,5);
    plot(pca_components(1,:), pca_components(2,:), strcat(mShapes(l+1),mColors(l+1))), hold on;
    subplot(6,5,10);
    plot(pca_components(2,:), pca_components(3,:), strcat(mShapes(l+1),mColors(l+1))), hold on;
    subplot(6,5,[3 4 8 9]);
    scatter3(pca_components(1,:), pca_components(2,:),pca_components(3,:), strcat(mShapes(l+1),
        mColors(l+1))), hold on;
end
subplot(6,5,5);
title('1st and 2nd PCA components');
xlabel('x^{pca}_1'), ylabel('x^{pca}_2'),
lgnd = legend('Class 0', 'Class 1', 'Class 2', 'Class 3');
lgnd.Location = 'bestoutside';
```

```
subplot(6,5,10);
title('2nd and 3rd PCA components');
xlabel('x^{pca}_2'), ylabel('x^{pca}_3'),
lgnd = legend('Class 0', 'Class 1', 'Class 2', 'Class 3');
lgnd.Location = 'bestoutside';

subplot(6,5,[3 4 8 9]);
title('Scatter Plot For all PCA components');
xlabel('x^{pca}_1'), ylabel('x^{pca}_2'), zlabel('x^{pca}_3'),
lgnd = legend('Class 0', 'Class 1', 'Class 2', 'Class 3');
lgnd.Location = 'bestoutside';

%
%% ========================= ERM Classification ========================= %%
lossMatrix = ones(L,L)-eye(L);
[avg_cost, p_error, decision_label] = ERMClassifyWithLlabels(x, labels, L, N, lossMatrix, priors,
    mu, Sigma);
display('Q2-A 0-1 loss matrix');
avg_cost
p_error

%% ========================= Confusion Matrix and P(error) ========================= %%
ConfusionMatrix = zeros(L,L);
for d = 0:L-1 % each decision option
    for l = 0:L-1 % each class label
        ind_dl = find(decision_label==d & labels==l);
        ConfusionMatrix(d+1,l+1) = sum(ind_dl)/sum(find(labels==l));
    end
end

%% ========================= Confusion Matrix Plot ========================= %%
% calculate the percentage accuracies
subplot(6,5,[14 15 19 20]),
plotConfusionMatrix(ConfusionMatrix, label_text);

%% ========================= Classification Scatter Plot With PCA ========================= %%
for l = 0:L-1 % each class label
    ind_l = find(labels==l);
    classification_result = (decision_label(ind_l) == l);
    correct_classifications = find(classification_result == 1);
    incorrect_classifications = find(classification_result == 0);
    x_label = x(:,labels==l);
    pca_components = pca(x_label, Sigma(:,:,l+1), mu(:, l+1));
    subplot(6,5,18),
    plot(pca_components(1,correct_classifications), pca_components(2,correct_classifications),
        strcat(mShapes(l+1),'g')), hold on
    plot(pca_components(1,incorrect_classifications), pca_components(2,incorrect_classifications),
        strcat(mShapes(l+1),'r')), hold on;
    subplot(6,5,13),
    plot(pca_components(2,correct_classifications), pca_components(3,correct_classifications),
        strcat(mShapes(l+1),'g')), hold on
    plot(pca_components(2,incorrect_classifications), pca_components(3,incorrect_classifications),
        strcat(mShapes(l+1),'r')), hold on;
    subplot(6,5,[11 12 16 17]),
    scatter3(x_label(1, correct_classifications), x_label(2, correct_classifications), x_label(3,
        correct_classifications), strcat(mShapes(l+1),'g')), hold on;
    scatter3(x_label(1, incorrect_classifications), x_label(2, incorrect_classifications), x_label
        (3, incorrect_classifications), strcat(mShapes(l+1),'r')), hold on;
%     subplot(6,5,,10),
%     plot(pca_components(1,correct_classifications), pca_components(3,correct_classifications),
    strcat(mShapes(l+1),'g')), hold on
%     plot(pca_components(1,incorrect_classifications), pca_components(3,incorrect_classifications)
    , strcat(mShapes(l+1),'r')), hold on;
end
subplot(6,5,18),
title('1st and 2nd PCA Classification');
xlabel('x^{pca}_1'), ylabel('x^{pca}_2'),
lgnd = legend('Class 0 Correcti', 'Class 0 Incorrect', ...
    'Class 1 Correcti', 'Class 1 Incorrect', ...
    'Class 2 Correcti', 'Class 2 Incorrect', ...
    'Class 3 Correcti', 'Class 3 Incorrect');
lgnd.Location = 'bestoutside';

subplot(6,5,13),
title('2nd and 3rd PCA Classification');
xlabel('x^{pca}_2'), ylabel('x^{pca}_3'),
lgnd = legend('Class 0 Correcti', 'Class 0 Incorrect', ...
    'Class 1 Correcti', 'Class 1 Incorrect', ...
    'Class 2 Correcti', 'Class 2 Incorrect', ...
    'Class 3 Correcti', 'Class 3 Incorrect');
lgnd.Location = 'bestoutside';


subplot(6,5,[11 12 16 17]),
title('Scatter Plot For all PCA components Classification');
xlabel('x^{pca}_1'), ylabel('x^{pca}_2'), zlabel('x^{pca}_3'),
lgnd = legend('Class 0 Correcti', 'Class 0 Incorrect', ...
    'Class 1 Correcti', 'Class 1 Incorrect', ...
    'Class 2 Correcti', 'Class 2 Incorrect', ...
    'Class 3 Correcti', 'Class 3 Incorrect');
lgnd.Location = 'bestoutside';
```

```
%% ======================== Loss Matrix Modification ======================== %%
lossMatrix = [0 1 2 3; 10 0 5 10; 20 10 0 1; 30 20 1 0];
[avg_cost, p_error, decision_label] = ERMClassifyWithLlabels(x, labels, L, N, lossMatrix, priors,
    mu, Sigma);
display('Q2-B loss matrix');
avg_cost
p_error

%% ======================== P(error) as a fn of seperation ======================== %%
sweep_assymmetry = 0;
sweep_seperation = 0.001:0.001:0.1;
N = 1000;
p_error_sweep = zeros(1,length(sweep_seperation));
p_error_sweep_pca = zeros(1,length(sweep_seperation));
lossMatrix = ones(L,L)-eye(L);

parfor i = 1:length(sweep_seperation)

    % set required seperations
    proximity = 1/sweep_seperation(i);
    lambda = repmat(proximity,1,L)+(0:L-1).*sweep_assymmetry;

    sigma1 = lambda(1).*eye(n);
    sigma2 = lambda(2).*eye(n);
    sigma3 = lambda(3).*eye(n);
    sigma4 = lambda(4).*eye(n);

    Sigmahat = cat(3, sigma1 ,sigma2, sigma3, sigma4);

    % generate data at required seperations

    [x, labels] = generateData4Gaussians(n, N, p, mu, Sigmahat);

    % estimate mean and covariance using sample mean and sample covariance

    x_label0 = x(:,labels==0);
    x_label1 = x(:,labels==1);
    x_label2 = x(:,labels==2);
    x_label3 = x(:,labels==3);

    mlabel = sort(labels, 'ascend');

    sigma1 = cov(x_label0');
    sigma2 = cov(x_label1');
    sigma3 = cov(x_label2');
    sigma4 = cov(x_label3');

    Sigmahat = cat(3, sigma1 ,sigma2, sigma3, sigma4);

    muhat1 = mean(x_label0, 2);
    muhat2 = mean(x_label1, 2);
    muhat3 = mean(x_label2, 2);
    muhat4 = mean(x_label3, 2);

    muhat = cat(2, muhat1, muhat2, muhat3, muhat4);

    % apply ERM with estimates

    [~, p_error_tmp, ~] = ERMClassifyWithLlabels(x, labels, L, N, lossMatrix, priors, muhat,
        Sigmahat);
    p_error_sweep(i) = p_error_tmp.*100;

    % apply pca to data and treat projections as new data

    pca_components_label0 = pca(x_label0);
    pca_components_label1 = pca(x_label1);
    pca_components_label2 = pca(x_label2);
    pca_components_label3 = pca(x_label3);

    pca_components = cat(2, pca_components_label0, pca_components_label1, pca_components_label2,
        pca_components_label3);

    sigma1 = cov(pca_components_label0');
    sigma2 = cov(pca_components_label1');
    sigma3 = cov(pca_components_label2');
    sigma4 = cov(pca_components_label3');

    Sigmahat = cat(3, sigma1 ,sigma2, sigma3, sigma4);

    muhat1 = mean(pca_components_label0, 2);
    muhat2 = mean(pca_components_label1, 2);
    muhat3 = mean(pca_components_label2, 2);
    muhat4 = mean(pca_components_label3, 2);

    muhat = cat(2, muhat1, muhat2, muhat3, muhat4);

    [~, p_error_tmp, ~] = ERMClassifyWithLlabels(pca_components, mlabel, L, N, lossMatrix, priors,
        muhat, Sigmahat);
    p_error_sweep_pca(i) = p_error_tmp.*100;
```

```
end

subplot(6,5, [21 22 23 24 25]), plot(sweep_seperation, p_error_sweep, 'm'), hold on,
plot(sweep_seperation, p_error_sweep_pca, 'r'),
title('P(error) as a function of class conditional proximity'),
xlabel('Seperation'), ylabel('P(error)'), legend('ERM with original Data', 'ERM with pca wightening');


%% ======================== P(error) as a fn of sweep_assymmetry ======================== %%
sweep_assymmetry = 0:1:99;
sweep_seperation = 0.1;
N = 1000;
p_error_sweep = zeros(1,length(sweep_assymmetry));
p_error_sweep_pca = zeros(1,length(sweep_seperation));

parfor i = 1:length(sweep_assymmetry)

    proximity = 1/sweep_seperation;
    lambda = repmat(proximity,1,L)+(0:L-1).*sweep_assymmetry(i);

    sigma1 = lambda(1).*eye(n);
    sigma2 = lambda(2).*eye(n);
    sigma3 = lambda(3).*eye(n);
    sigma4 = lambda(4).*eye(n);

    Sigmahat = cat(3, sigma1 ,sigma2, sigma3, sigma4);

    [x, labels] = generateData4Gaussians(n, N, p, mu, Sigmahat);
 % estimate mean and covariance using sample mean and sample covariance

    x_label0 = x(:,labels==0);
    x_label1 = x(:,labels==1);
    x_label2 = x(:,labels==2);
    x_label3 = x(:,labels==3);

    mlabel = sort(labels, 'ascend');

    sigma1 = cov(x_label0');
    sigma2 = cov(x_label1');
    sigma3 = cov(x_label2');
    sigma4 = cov(x_label3');

    Sigmahat = cat(3, sigma1 ,sigma2, sigma3, sigma4);

    muhat1 = mean(x_label0, 2);
    muhat2 = mean(x_label1, 2);
    muhat3 = mean(x_label2, 2);
    muhat4 = mean(x_label3, 2);

    muhat = cat(2, muhat1, muhat2, muhat3, muhat4);

    % apply ERM with estimates

    [~, p_error_tmp, ~] = ERMClassifyWithLlabels(x, labels, L, N, lossMatrix, priors, mu, Sigmahat);
    p_error_sweep(i) = p_error_tmp.*100;

    pca_components_label0 = pca(x_label0);
    pca_components_label1 = pca(x_label1);
    pca_components_label2 = pca(x_label2);
    pca_components_label3 = pca(x_label3);

    pca_components = cat(2, pca_components_label0, pca_components_label1, pca_components_label2, pca_components_label3);

    sigma1 = cov(pca_components_label0');
    sigma2 = cov(pca_components_label1');
    sigma3 = cov(pca_components_label2');
    sigma4 = cov(pca_components_label3');

    Sigmahat = cat(3, sigma1 ,sigma2, sigma3, sigma4);

    muhat1 = mean(pca_components_label0, 2);
    muhat2 = mean(pca_components_label1, 2);
    muhat3 = mean(pca_components_label2, 2);
    muhat4 = mean(pca_components_label3, 2);

    muhat = cat(2, muhat1, muhat2, muhat3, muhat4);

    [~, p_error_tmp, ~] = ERMClassifyWithLlabels(pca_components, mlabel, L, N, lossMatrix, priors, muhat, Sigmahat);
    p_error_sweep_pca(i) = p_error_tmp.*100;
end

subplot(6,5, [26 27 28 29 30]), plot(sweep_assymmetry, p_error_sweep, 'm'), hold on,
plot(sweep_assymmetry, p_error_sweep_pca, 'r');
title('P(error) as a function of class conditional assymmetry'),
xlabel('Assymmetry'), ylabel('P(error)'), legend('ERM with original Data', 'ERM with pca wightening');

%% ======================== Helper Functions ======================== %%
```

```
function [avg_cost, p_error, decision_label] = ERMClassifyWithLlabels(x, labels, L, N, lossMatrix,
    priors, mu, Sigma)

p_x_given_l = zeros(L,N);
for l = 0:L-1
    p_x_given_l(l+1,:) = evalGaussian(x, mu(:,l+1), Sigma(:,:,l+1));
end
p_x = priors*p_x_given_l; % For minimization I probably don't need this because it's just a scaling
     factor
p_l_given_x = p_x_given_l.*repmat(priors',1,N)./repmat(p_x,L,1);
expectedRisks = lossMatrix*p_l_given_x;
[~, decision_label] = min(expectedRisks, [], 1);
decision_label = decision_label - 1;

costs = zeros(1,N);
for i = 1:N
    costs(i) = lossMatrix(decision_label(i)+1, labels(i)+1);
end

avg_cost = sum(costs)/length(costs);

p_error = length(find(decision_label ~= labels))/length(labels);

end

function [x, labels] = generateData4Gaussians(n, N, p, mu, Sigma)

labels = zeros(1,N);
x = zeros(n,N); % save up space
for i = 1:N
    random_number = rand(1,1);
    if random_number >= p(3)
        labels(i) = 3;
        x(:,i) = mvnrnd(mu(:,4),Sigma(:,:,4));
    elseif random_number >= p(2)
        labels(i) = 2;
        x(:,i) = mvnrnd(mu(:,3),Sigma(:,:,3));
    elseif random_number >= p(1)
        labels(i) = 1;
        x(:,i) = mvnrnd(mu(:,2),Sigma(:,:,2));
    else
        labels(i) = 0;
        x(:,i) = mvnrnd(mu(:,1),Sigma(:,:,1));
    end
end
end

function [] = plotConfusionMatrix(ConfusionMatrix, labels)

L = length(labels);

confpercent = ConfusionMatrix.*100;

% plotting the colors
imagesc(confpercent),
title('Confusion Matrix');
ylabel('Decision'); xlabel('Label');

% set the colormap
colormap(flipud(gray));

% Create strings from the matrix values and remove spaces
textStrings = strcat(num2str(confpercent(:), '%.1f\n'),'%');
textStrings = strtrim(cellstr(textStrings));

% Create x and y coordinates for the strings and plot them
[x,y] = meshgrid(1:L);
hStrings = text(x(:),y(:),textStrings(:), ...
    'HorizontalAlignment','center');

% Get the middle value of the color range
midValue = mean(get(gca,'CLim'));

% Choose white or black for the text color of the strings so
% they can be easily seen over the background color
textColors = repmat(confpercent(:) > midValue,1,3);
set(hStrings,{'Color'},num2cell(textColors,2));

% Setting the axis labels
set(gca,'XTick',1:L,...
    'XTickLabel',labels,...
    'YTick',1:L,...
    'YTickLabel',labels,...
    'TickLength',[0 0]);

end

function g = evalGaussian(x,mu,Sigma)
% Evaluates the Gaussian pdf N(mu,Sigma) at each coumn of X
[n,N] = size(x);
C = ((2*pi)^n * det(Sigma))^(-1/2);
E = -0.5*sum((x-repmat(mu,1,N)).*(inv(Sigma)*(x-repmat(mu,1,N))),1);
```

```
g = C*exp(E);
end

function [y] = pca(x, sigma, mu)

% If original distribution parameters not passed then
% use sample-based estimates of mean and covariance matrix
if ~exist('sigma', 'var')
    Sigmahat = cov(x');
else
    Sigmahat = sigma;
end

if ~exist('mu', 'var')
    muhat = mean(x,2)';
else
    muhat = mu';
end

% make data 0 mean
xzm = x - muhat*ones(size(x));

% Get the eigenvectors (in Q) and eigenvalues (in D) of the
% estimated covariance matrix
[Q,D] = eig(Sigmahat);

% Sort the eigenvalues from large to small, reorder eigenvectors
% accordingly as well.
[d,ind] = sort(diag(D),'descend');
Q = Q(:,ind);
D = diag(d);

% Calculate the principal components (in y)
% Also whiten components so their norms are 1
y = D^(-1/2)*Q'*xzm;
end
```