EECE 5644 Assignment 1

Name: Liangshe Li
NUID: 001586357

Question1

The probability density function (pdf) for a 4-dimensional real-valued random vector X is as follows: p(x) = p(x|L = 0)P(L = 0)+ p(x|L = 1)P(L = 1). Here L is the true class label that indicates which class-label-conditioned pdf generates the data.

The class priors are P(L = 0) = 0.7 and P(L = 1) = 0.3. The class class-conditional pdfs are p(x|L = 0) = g(x|m0,C0) and p(x|L = 1) = g(x|m1,C1), where g(x|m,C) is a multivariate Gaussian probability density function with mean vector m and covariance matrix C. The parameters of the class-conditional Gaussian pdfs are:

$$\mathbf{m}_0 = \begin{bmatrix} -1 \\ -1 \\ -1 \\ -1 \end{bmatrix} \quad \mathbf{C}_0 = \begin{bmatrix} 2 & -0.5 & 0.3 & 0 \\ -0.5 & 1 & -0.5 & 0 \\ 0.3 & -0.5 & 1 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix} \quad \mathbf{m}_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad \mathbf{C}_1 = \begin{bmatrix} 1 & 0.3 & -0.2 & 0 \\ 0.3 & 2 & 0.3 & 0 \\ -0.2 & 0.3 & 1 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix}$$

For numerical results requested below, generate 10000 samples according to this data distribution, keep track of the true class labels for each sample. Save the data and use the same data set in all cases.

Part A: ERM classification using the knowledge of true data pdf:

1.Specify the minimum expected risk classification rule in the form of a likelihood-ratio test: $\frac{P(x|L=1)}{P(x|L=0)} > \gamma$, where the threshold γ is a function of class priors and fixed (nonnegative) loss values for each of the four cases D = i|L = j where D is the decision label that is either 0 or 1, like L.

2. Implement this classifier and apply it on the 10K samples you generated. Vary the threshold γ gradually from 0 to ∞, and for each value of the threshold compute the true positive (detection) probability P(D = 1|L = 1; γ) and the false positive (false alarm) probability P(D = 1|L = 0; γ). Using these paired values, trace/plot an approximation of the ROC curve of the minimum expected risk classifier. Note that at γ = 0 The ROC curve should be at (11), and as gamma increases it should traverse towards (00). Due to the finite number of samples used to estimate probabilities, your ROC curve approximation should reach this destination value for a finite threshold value. Keep track of (D = 0|L = 1; γ) and P(D = 1|L = 0; γ) values for each

gamma value for use in the next section.

3. Determine the threshold value that achieves minimum probability of error, and on the ROC curce, superimpose clearly (using a different color/shape marker) the true positive and false positive values attained by this minimum-P(error) classifier. Calculate and report an estimate of the minimum probability of error that is achievable for this data distribution. Note that P(error; γ) = P(D = 1|L = 0; γ)P(L = 0)+P(D = 0|L = 1; γ)P(L = 1). How does your empirically selected γ value that minimizes P(error) compare with the theoretically optimal threshold you compute from priors and loss values?

Part B: ERM classification attempt using incorrect knowledge of data distribution (Naive Bayesian Classifier, which assumes features are independent given each class label)... For this part, assume that you know the true class prior probabilities, but for some reason you think that the class conditional pdfs are both Gaussian with the true means, but (incorrectly) with covariance matrices that are diagonal (with diagonal entries equal to true variances, off-diagonal entries equal to zeros, consistent with the independent feature assumption of Naive Bayes). Analyze the impact of this model mismatch in this Naive Bayesian (NB) approach to classifier design by repeating the same steps in Part A on the same 10K sample data set you generated earlier. Report the same results, answer the same questions. Did this model mismatch negatively impact your ROC curve and minimum achievable probability of error?

Part C: In the third part of this exercise, repeat the same steps as in the previous two cases, but this time using a Fisher Linear Discriminant Analysis (LDA) based classifier. Using the 10Kavailable samples, estimate the class conditional pdf mean and covariance matrices using sample average estimators for mean and covariance. From these estimated mean vectors and covariance matrices, determine the Fisher LDA projection weight vector (via the generalized eigendecomposition of within and between class scatter matrices): W_{LDA}. For the classification rule $W_{LDA}^T X$ compared to a threshold τ, which takes values from $-\infty$ to ∞, trace the ROC curve. Identify the threshold at which the probability of error (based on sample count estimates) is minimized, and clearly mark that operating point on the ROC curve estimate. Discuss how this LDA classifier performs relative to the previous two classifiers.

Note: When finding the Fisher LDA projection matrix, do not be concerned about the differencein the class priors. When determining the between-class and within-class scatter matrices, useequal weights for the class means and covariances, like we did in class.

Answer:

Part A

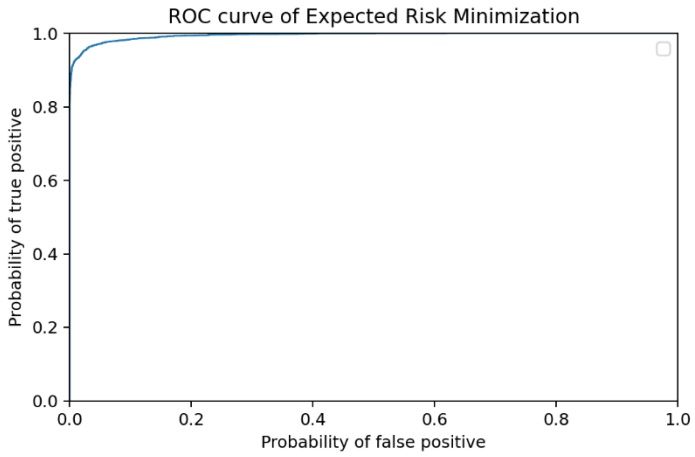
1. The minimum expected risk classification rule in the form of a likelihood-ratio test is:

$$\frac{P(x|L=1)}{P(x|L=0)} \lessgtr \frac{(\lambda_{10}-\lambda_{00})P(L=0)}{(\lambda_{01}-\lambda_{11})P(L=1)} = \gamma$$

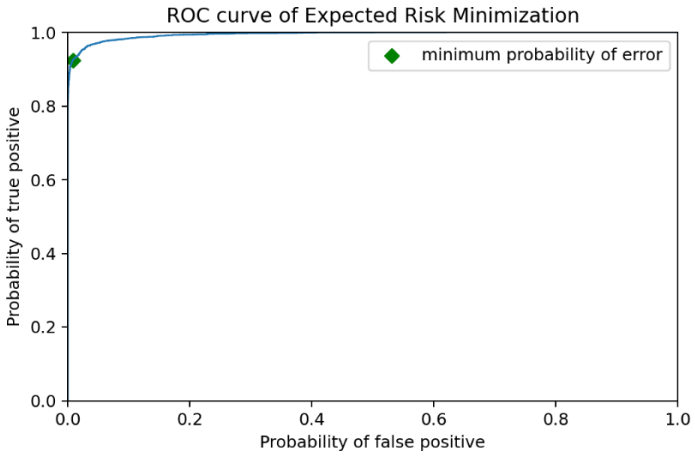
If the classification is wrong, the loss threshold = 1, so we can get that

$$(D(x)=1)\frac{P(x|L=1)}{P(x|L=0)} > \frac{(1-0)*0.7}{(1-0)*0.3} = \gamma \approx 2.333$$

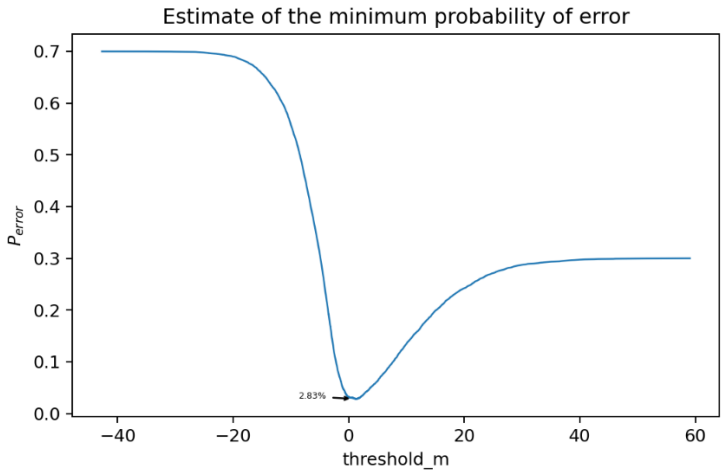
2. The ROC of the minimum expected risk classifier is shown as below:



3. First, I add one green mark which traces the classifier of the minimum error according to the last result, as shown below:



The picture below shows the relationship between the selected γ and the probability of error:

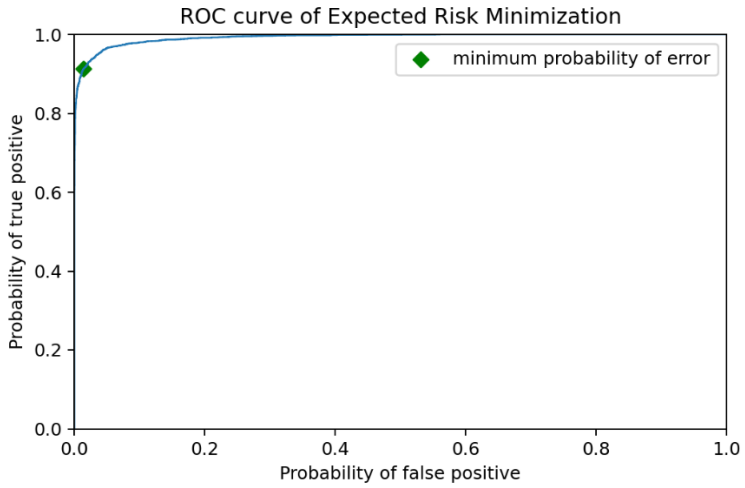


I find that when $\gamma = 1.298$, we can get the minimized probability of error which is 2.83%. The theoretically optimal threshold is 2.333. The estimated threshold value and the theoretical

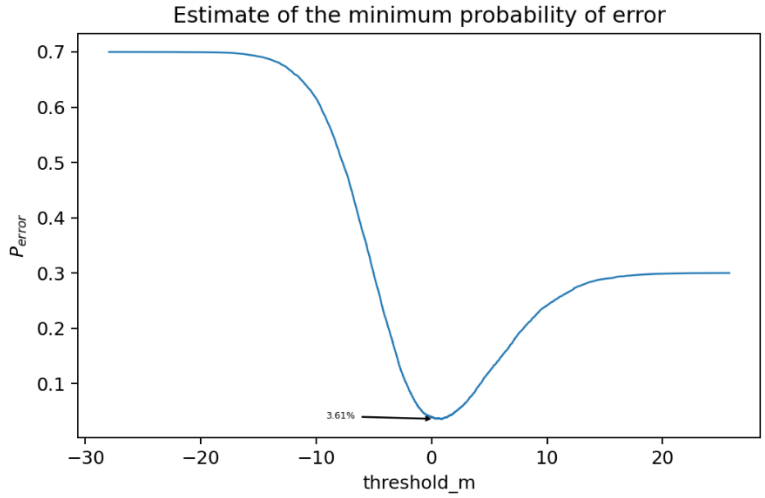
threshold value are quite different, but it is reasonable because the data we use is generated randomly.

Part B

With the diagonal covariance matrices, I get the plot of ROC curve as follows:



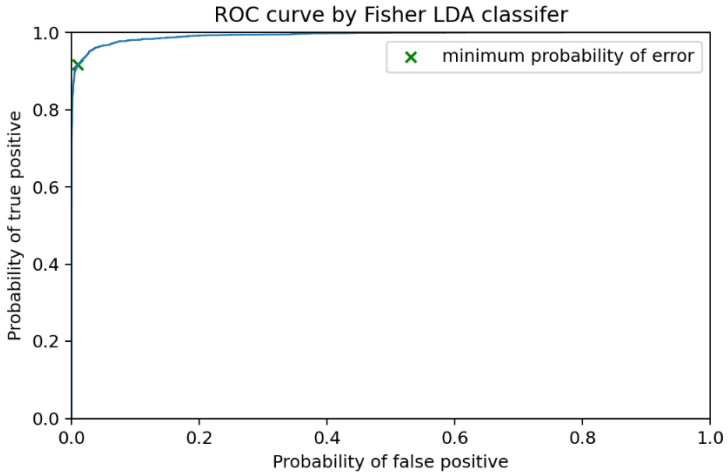
And I can get the relationship between threshold and probability of error as the plot below shows:



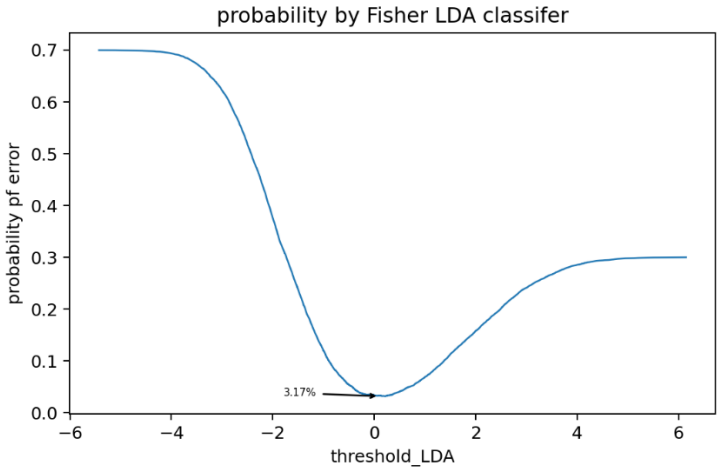
From the above results, the ROC curve is almost not impacted, however, the probability of error is increased. If we use the diagonal covariance matrices, the minimum probability of error is 3.61%, higher than the true class (2.83%) in part A.

Part C

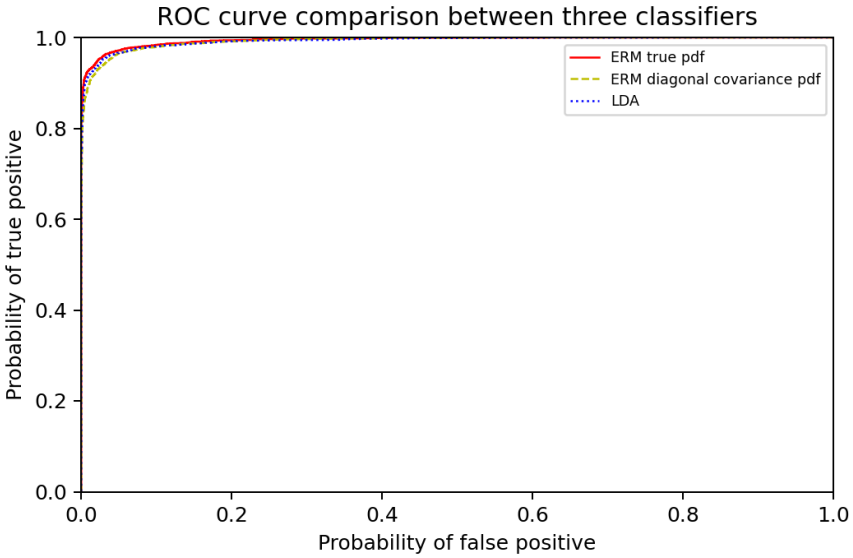
We get the ROC curve using Fisher LDA classification as following:

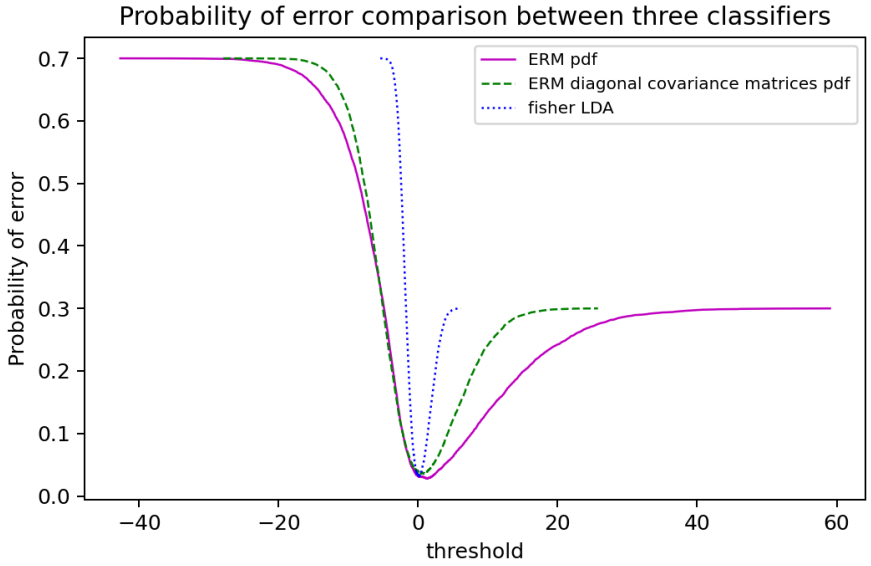


The relationship between threshold and probability of error is below:



We can find that the minimum probability of error is 3.17%.
To compare the performance of LDA classifier with other two classifiers, we plot the following two graphs to help us analyze:





From the last two plots, we can find that Expected Risk Minimization classifier using true class has best performance. Fisher LDA classifier does not have better performance than ERM using mismatch class in most of time, but the minimize probability of error using LDA is lower than ERM using mismatch class.

Question2

A 3-dimensional random vector X takes values from a mixture of four Gaussians. One of these Gaussians represent the class-conditional pdf for class 1, and another Gaussian represents the classconditional pdf for class 2. Class 3 data originates from a mixture of the remaining 2 Gaussian components with equal weights. For this setting where labels $L \in \{1,2,3\}$, pick your own classconditional pdfs $p(x|L = j)$, $j \in \{1,2,3\}$ as described. Try to approximately set the distances between means of pairs of Gaussians to approximately 2 to 3 times the average standard deviation of the Gaussian compoenents, so that there is some significant overlap between class-conditional pdfs. Set class priors to 0.3,0.3,0.4.

Part A: Minimum probability of error classification (0-1 loss, also referred to as Bayes Decision rule or MAP classifier).

1. Generate 10000 samples from this data distribution and keep track of the true labels of each sample.
2. Specify the decision rule that achieves minimum probability of error (i.e., use 0-1 loss), implement this classifier with the true data distribution knowledge, classify the 10K samples and count the samples corresponding to each decision-label pair to empirically estimate the confusion matrix whose entries are $P(D = i|L = j)$ for $i, j \in \{1,2,3\}$.

3. Provide a visualization of the data (scatter-plot in 3-dimensional space), and for each sample indicate the true class label with a different marker shape (dot, circle, triangle, square) and whether it was correctly (green) or incorrectly (red) classified with a different marker color as indicated in parentheses.

Part B: Repeat the exercise for the ERM classification rule with the following loss matrices which respectively care 10 times or 100 times more about not making mistakes when L = 3:

$$\Lambda_{10} = \begin{bmatrix} 0 & 1 & 10 \\ 1 & 0 & 10 \\ 1 & 1 & 0 \end{bmatrix} \quad and \quad \Lambda_{100} = \begin{bmatrix} 0 & 1 & 100 \\ 1 & 0 & 100 \\ 1 & 1 & 0 \end{bmatrix} \tag{1}$$

Note that, the $(i,j)^{th}$ entry of the loss matrix indicates the loss incurred by deciding on class i when the true label is j. For this part, using the 10K samples, estimate the minimum expected risk that this optimal ERM classification rule will achieve. Present your results with visual and numerical reprentations. Briefly discuss interesting insights, if any.

Answer:

The 4 Gaussians we set is as following:

$$\mu_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \quad \Sigma_1 = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 3 \end{bmatrix}, \quad p(L=1) = 0.3$$

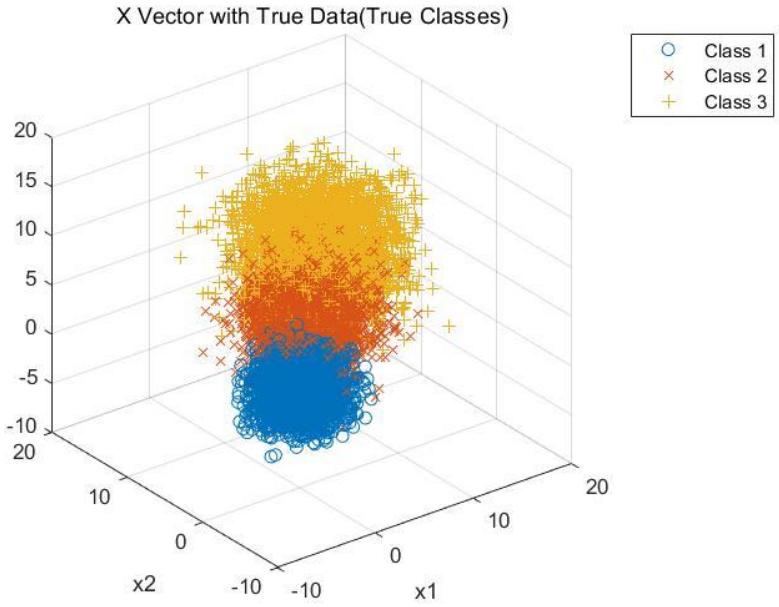
$$\mu_2 = \begin{bmatrix} 4 \\ 4 \\ 4 \end{bmatrix}, \quad \Sigma_2 = \begin{bmatrix} 6 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 6 \end{bmatrix}, \quad p(L=2) = 0.3$$

$$\mu_{31} = \begin{bmatrix} 8 \\ 8 \\ 8 \end{bmatrix}, \quad \Sigma_{31} = \begin{bmatrix} 9 & 0 & 0 \\ 0 & 9 & 0 \\ 0 & 0 & 9 \end{bmatrix}$$

$$\mu_{32} = \begin{bmatrix} 12 \\ 12 \\ 12 \end{bmatrix}, \quad \Sigma_{32} = \begin{bmatrix} 12 & 0 & 0 \\ 0 & 12 & 0 \\ 0 & 0 & 12 \end{bmatrix} \quad , p(L=3) = 0.4$$

Part A

1. The 10000 samples we generate is in the picture below(using a different shape to show):



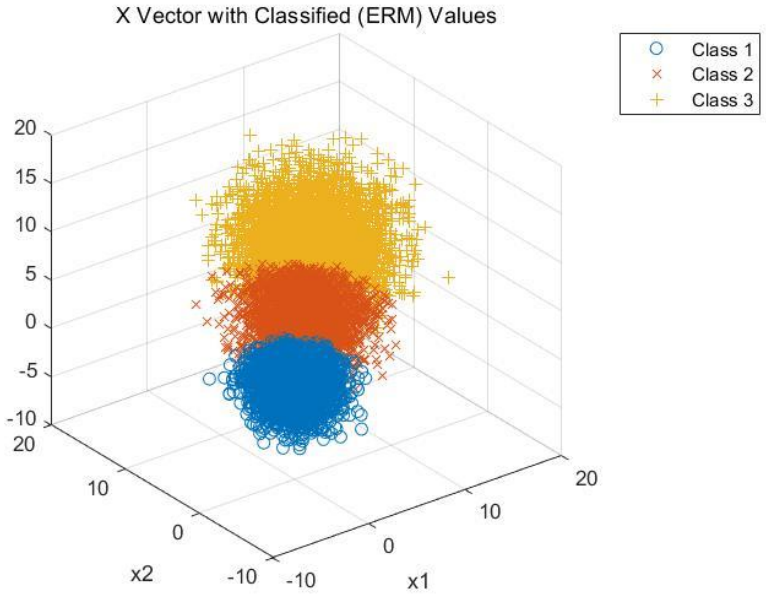
2. I use ERM as my decision rule, which is:

$$D(x) = \underset{D \in :1,2,3,4\}}{arg\min} \sum_{l=1}^{\breve{C}} \lambda_{Dl}\, p(L = l|x)$$

where C is the number of classes

λ_{Dl} is the loss for classifying a point from label l in class D

And I get the classified values as follows:



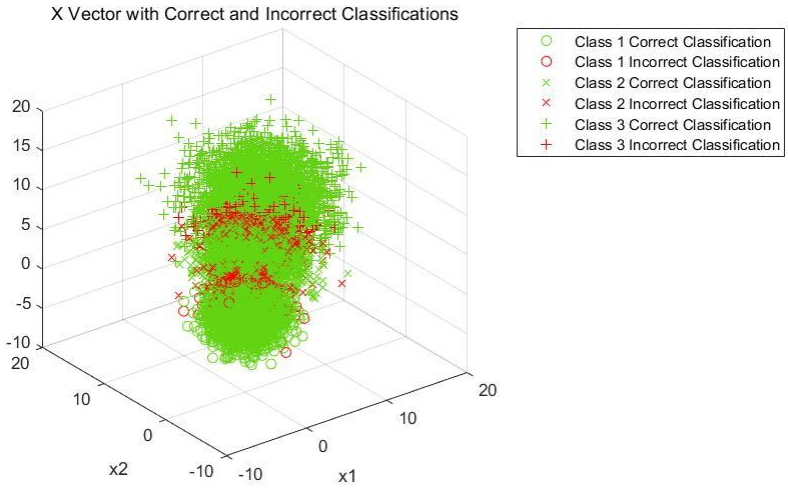
The loss matrix I use is shown as following:

$$\Lambda = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

The confusion matrix I estimate is shown in the table below as the probability of classification for each class:

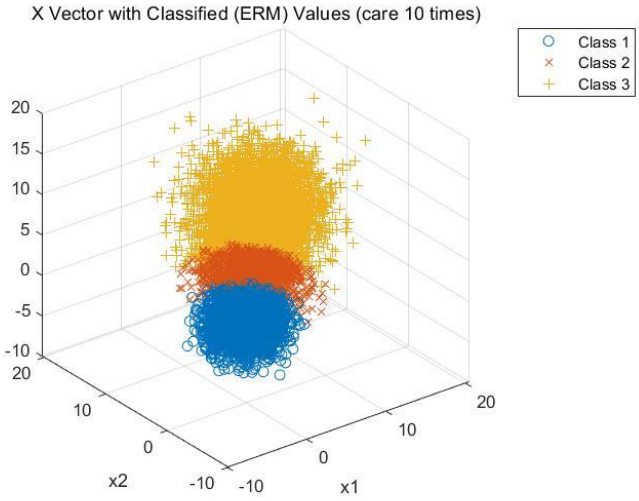
		Truth		
		1	2	3
Decision	1	0.9616	0.0541	0
	2	0.0384	0.8345	0.1001
	3	0	0.1114	0.8999

3. The picture below shows whether the classification is correct (in green) or incorrect(in red):

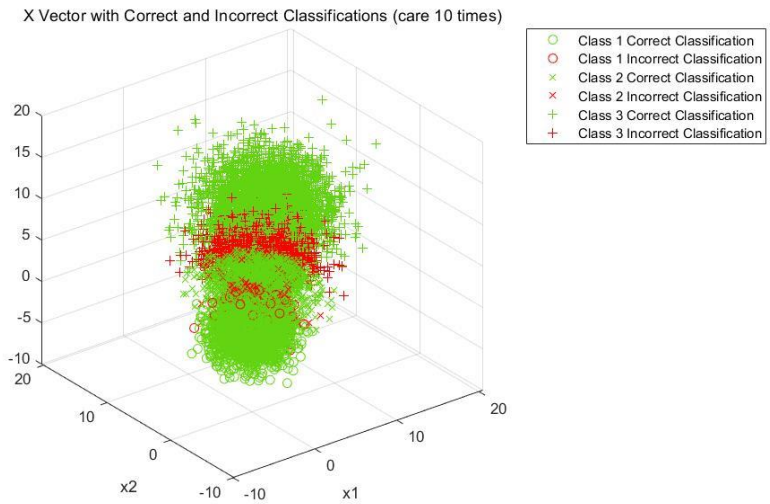


Part B

The result I get using ERM classification which care 10 times about not making mistakes when L=3 is shown in the following graph:



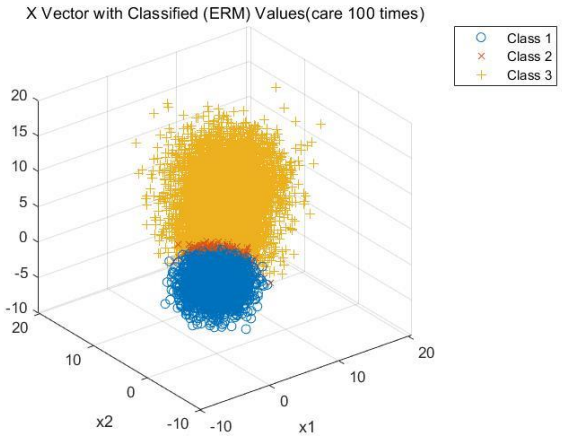
Like in part A, the correct and incorrect classification are shown as below (red is correct, green is incorrect):



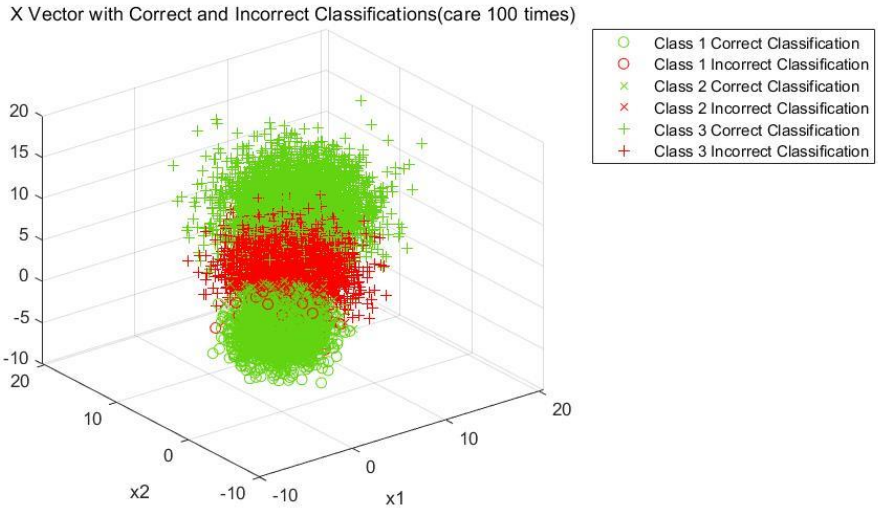
The resulting confusion matrix is shows in the following table:

		Truth		
		1	2	3
Decision	1	0.9616	0.0541	0
	2	0.0384	0.5199	0.0166
	3	0	0.4261	0.9834

The result I get using ERM classification which care 100 times about not making mistakes is shown as below:



The correct and incorrect classification are shown as below (red is correct, green is incorrect):



The resulting confusion matrix is shows in the following table:

		Truth		
		1	2	3
Decision	1	0.9616	0.0541	0
	2	0.0323	0.0771	0
	3	0.0061	0.8689	0.9998

Question3

Download the following datasets...

• Wine Quality dataset located at https://archive.ics.uci.edu/ml/datasets/Wine+Quality consists of 11 features, and class labels from 0 to 10 indicating wine quality scores. There are 4898 samples.

• Human Activity Recognition dataset located at https://archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones consists of 561 features, and 6 activity labels. There are 10299 samples.

Implement minimum-probability-of-error classifiers for these problems, assuming that the class conditional pdf of features for each class you encounter in these examples is a Gaussian. Using all available samples from a class, with sample averages, estimate mean vectors and covariance matrices. Using sample counts, also estimate class priors. In case your sample estimates of covariance matrices are ill-conditioned, consider adding a regularization term to your covariance estimate as in: $C_{\text{Regularized}} = C_{\text{SampleAverage}} + \lambda I$ where $\lambda > 0$ is a small regularization parameter that ensures the regularized covariance matrix CRegularized has all eigenvalues larger than this parameter.

With these estimated (trained) Gaussian class conditional pdfs and class priors, apply the minimum-P(error) classification rule on all (training) samples, count the errors, and report the error probability estimate you obtain for each problem. Also report the confusion matrices for both datasets, for this classification rule.

Visualize the datasets in various 2 or 3 dimensional projections (either subsets of features, or if you know how to do it, using principal component analyses). Discuss if Gaussian class conditional models are appropriate for these datasets and how your model choice might have influenced the confusion matrix and probability of error values you obtained in the experiments conducted above. Make sure you explain in rigorous detail what your modeling assumptions are, how you estimated/selected necessary parameters for your model and classification rule, and describe your analyses in mathematical terms supplemented by numerical and visual results in a way that conveys your understanding of what you have accomplished and demonstrated.

Hint: Later in the course, we will talk about how to select regularization/hyper-parameters. For now, you may consider using a value on the order of arithmetic average of sample covariance matrix estimate non-zero $\lambda = \alpha trace(C_{SampleAverage})/rank(C_{SampleAverage})$ *or geometric average of sample covariance matrix estimate non-zero eigenvalues, where* $0<\alpha<1$ *is a small real number. This makes your regularization term proportional to the eigenvalues observed in the sample covariance estimate.*

Answer:

I use the LDA classifier to solve these problems.

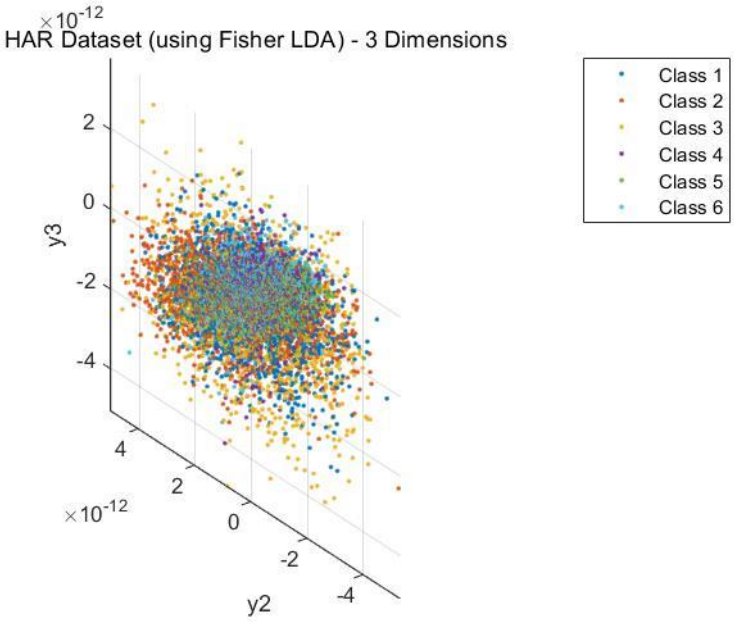
For Human Activity Recognition dataset, the estimated error for each class is in the following table:

	Class1	Class2	Class3	Class4	Class5	Class6
P(error)	0.168	0.160	0.143	0.167	0.181	0.182

The confusion matrix I get is below:

	1	2	3	4	5	6
1	0.1683068	0	0	0	0	0
2	0	0.1598236	0.0006787	0	0	0
3	0	0	0.1418392	0	0	0
4	0	0	0	0.1659315	0.0003393	0
5	0	0	0	0.0006787	0.1801832	0
6	0	0	0	0	0	0.1822192

The projection of the dataset is in the picture below:



In this classifier, I add a regularization term to my estimate:

$$C_{Regularized} = C_{SampleAverage} + \lambda I$$

And I set the regularized lambda=0.00001.

To minimize the probability of error using Fisher LDA classification(if 2 classes), we need to calculate:

$$\max_{\omega} \frac{(\mu_1 - \mu_2)^2}{\sigma_1^2 + \sigma_2^2}$$

so that we can make all data "Maximally separated".

In this way, we could find that the data we estimate in the resulting graph above is not so separated, so there is some problems with our classifier, and the probability of error for each class is very high, so the LDA classification might not be suitable for this dataset. The next step is to use other classification (e.g. ERM) to solve the problem.

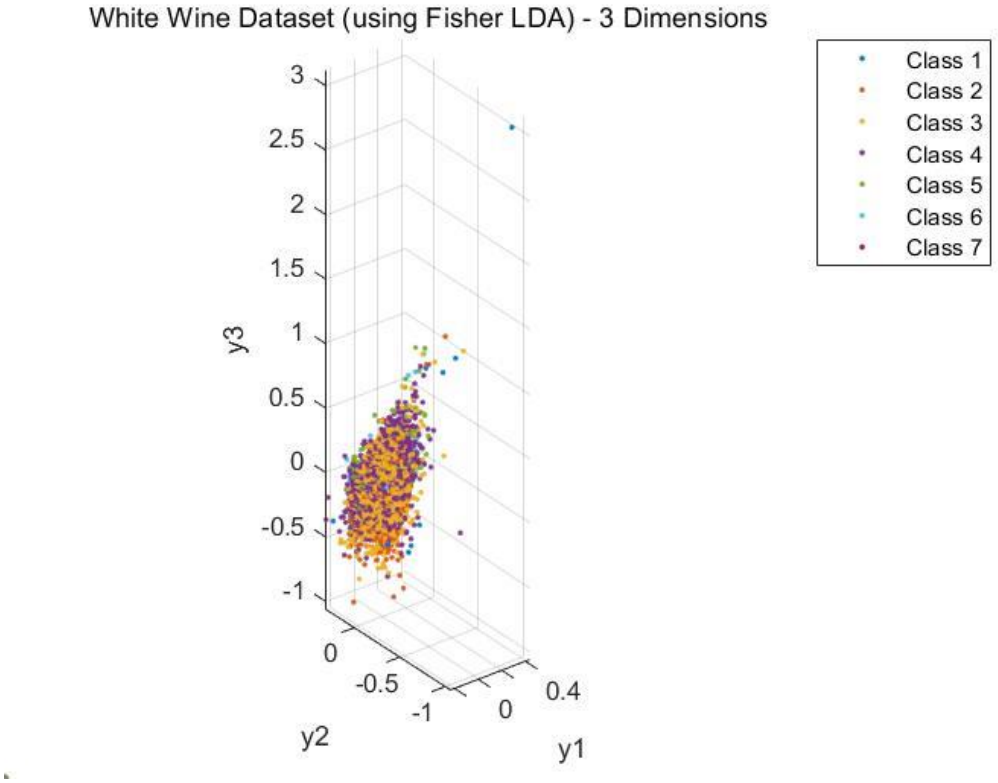
I also use the LDA classification to solve the problem of Wine Quality, the probability of error for each class I get is below:

	Class1	Class2	Class3	Class4	Class5	Class6	Class7
P(error)	0.004	0.033	0.297	0.449	0.180	0.036	0.001

The confusion matrix I get is:

	1	2	3	4	5	6	7
1	0.006	0.002	0.004	0	0	0	0
2	0	0.033	0	0	0	0	0
3	0	0.001	0.3	0.032	0	0	0
4	0	0	0	0.499	0.001	0	0
5	0	0	0	0.001	0.192	0	0
6	0	0	0	0	0	0.036	0
7	0	0	0	0	0	0	0.001

The projection of dataset is:



Compared with the two results, I find that LDA classification might have better performance when the size of dataset is small(like <10000 samples), this is why in Wine Quality(4898 samples) the probability of error is quite smaller than which in Human Activity Recognition, so when we need to process dataset which is in a large size, we need to consider other types of classification, because LDA classification might not have performance as expected.

Codes

Qustion1(Python)

```
import numpy as np
import numpy.random as ran
from scipy import linalg
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal

# make data fixed everytime generated
ran.seed(666)

# set the classifier
classifier = np.array([.7, .3])
class_t = classifier.size

# set the mean vector
mean = np.array([-1 * np.ones((4, 1)), np.ones((4, 1))])

# set the covariance
covariance = np.array([[[2, -.5, .3, 0],
                        [-.5, 1, -.5, 0],
                        [.3, -.5, 1, 0],
                        [0, 0, 0, 2]],

                       [[1, .3, -.2, 0],
                        [.3, 2, .3, 0],
                        [-.2, .3, 1, 0],
                        [0, 0, 0, 3]]])

# set the sample size
n = 10000
ran_num = ran.rand(n)

# data structure and generate data
true_x = np.zeros((mean.shape[1], n))

class_sample_total = np.zeros(class_t).astype(int)
temp1 = np.insert(np.cumsum(classifier), 0, 0)

true_data = ran_num > classifier[0]

for i in range(class_t):
    judge = (ran_num >= temp1[i]) & (ran_num < temp1[i + 1])
```

```
        class_sample_total[i] = np.sum(judge)
        true_x[:, judge] = ran.multivariate_normal(mean[i, :, :].flatten(), covariance[i, :, :],
class_sample_total[i]).T

# ------ Part A ------

eps = 1e-16

divide_result = np.zeros(n)

for i in range(n):
        divide_result[i] = np.log(multivariate_normal.pdf(true_x[:, i],
mean=mean[1, :, :].flatten(), cov=covariance[1, :, :]) /
                                  multivariate_normal.pdf(true_x[:, i],
mean=mean[0, :, :].flatten(), cov=covariance[0, :, :]))

# select threshold (mid point)
divide_sort = np.sort(divide_result)
threshold_m = (divide_sort[0:-1] + divide_sort[1:]) / 2
threshold_m = np.insert(threshold_m, 0, divide_sort[0] - eps)
threshold_m = np.append(threshold_m, divide_sort[-1] + eps)

P_01 = np.zeros(threshold_m.size)
P_10 = np.zeros(threshold_m.size)
P_11 = np.zeros(threshold_m.size)
Perr_a = np.zeros(threshold_m.size)

for i in range(threshold_m.size):
        ERM_decision = divide_result > threshold_m[i]
        P_01[i] = np.sum(~ERM_decision & true_data) / np.sum(true_data)
        P_10[i] = np.sum(ERM_decision & ~true_data) / np.sum(~true_data)
        P_11[i] = np.sum(ERM_decision & true_data) / np.sum(true_data)
        Perr_a[i] = P_01[i] * classifier[1] + P_10[i] * classifier[0]

threshold_best = threshold_m[np.argmin(Perr_a)]

print(threshold_best)

# draw the graph of ROC curve
figure1 = plt.figure(figsize=[6, 4], dpi=180)
firstP = figure1.add_subplot(111)
firstP.plot(P_10, P_11, linewidth=1)

# make the curve begin from (0,0) to (1,1)
```

```
firstP.set_xlim([0, 1])
firstP.set_ylim([0, 1])

firstP.set_xlabel('Probability of false positive')
firstP.set_ylabel('Probability of true positive')
firstP.set_title('ROC curve of Expected Risk Minimization')

plt.tight_layout()
plt.savefig('1_A1.png')
plt.show()

figure1 = plt.figure(figsize=[6, 4], dpi=180)
firstP = figure1.add_subplot(111)
firstP.plot(P_10, P_11, linewidth=1)

# mark the minimum probability of error
firstP.scatter(P_10[np.argmin(Perr_a)], P_11[np.argmin(Perr_a)], c='g',
                   marker='D', label='minimum probability of error')

# make the curve begin from (0,0) to (1,1)
firstP.set_xlim([0, 1])
firstP.set_ylim([0, 1])

firstP.set_xlabel('Probability of false positive')
firstP.set_ylabel('Probability of true positive')
firstP.set_title('ROC curve of Expected Risk Minimization')
firstP.legend()

plt.tight_layout()
plt.savefig('1_A2.png')
plt.show()

# estimated minimized probability of error
figure2 = plt.figure(figsize=[6, 4], dpi=180)
secondP = figure2.add_subplot(111)
secondP.plot(threshold_m, Perr_a, linewidth=1)
plt.annotate(f'{min(Perr_a):4.2%}', fontsize=5,
                     xy=(threshold_best - .5, min(Perr_a)),
                     xytext=(threshold_best - 10, min(Perr_a)),
                     arrowprops=dict(arrowstyle='->'))
secondP.set_xlabel('threshold_m')
secondP.set_ylabel(r'$P_{error}$')
secondP.set_title('Estimate of the minimum probability of error')
plt.tight_layout()
```

```
plt.savefig('1_A3.png')
plt.show()

# ------ Part B ------
divide_result = np.zeros(n)

for i in range(n):
    divide_result[i] = np.log(multivariate_normal.pdf(true_x[:, i], mean=mean[1, :, :].flatten(), cov=np.diag(np.diag(covariance[1, :, :])))) / multivariate_normal.pdf(true_x[:, i], mean=mean[0, :, :].flatten(), cov=np.diag(np.diag(covariance[0, :, :]))))

# select threshold (mid point)
divide_sort = np.sort(divide_result)
threshold_m2 = (divide_sort[0:-1] + divide_sort[1:]) / 2
threshold_m2 = np.insert(threshold_m2, 0, divide_sort[0] - eps)
threshold_m2 = np.append(threshold_m2, divide_sort[-1] + eps)

P_01_m = np.zeros(threshold_m2.size)
P_10_m = np.zeros(threshold_m2.size)
P_11_m = np.zeros(threshold_m2.size)
Perr_m = np.zeros(threshold_m2.size)

for i in range(threshold_m2.size):
    ERM_decision = divide_result > threshold_m2[i]
    P_01_m[i] = np.sum(~ERM_decision & true_data) / np.sum(true_data)
    P_10_m[i] = np.sum(ERM_decision & ~true_data) / np.sum(~true_data)
    P_11_m[i] = np.sum(ERM_decision & true_data) / np.sum(true_data)
    Perr_m[i] = P_01_m[i] * classifier[1] + P_10_m[i] * classifier[0]

threshold_best = threshold_m2[np.argmin(Perr_m)]

figure1 = plt.figure(figsize=[6, 4], dpi=180)
firstP = figure1.add_subplot(111)
firstP.plot(P_10_m, P_11_m, linewidth=1)

# mark the minimum probability of error
firstP.scatter(P_10_m[np.argmin(Perr_m)], P_11_m[np.argmin(Perr_m)], c='g',
               marker='D', label='minimum probability of error')

# make the curve begin from (0,0) to (1,1)
firstP.set_xlim([0, 1])
firstP.set_ylim([0, 1])
```

```
firstP.set_xlabel('Probability of false positive')
firstP.set_ylabel('Probability of true positive')
firstP.set_title('ROC curve of Expected Risk Minimization')
firstP.legend()

plt.tight_layout()
plt.savefig('1_A4.png')
plt.show()

# estimated minimized probability of error
figure2 = plt.figure(figsize=[6, 4], dpi=180)
secondP = figure2.add_subplot(111)
secondP.plot(threshold_m2, Perr_m, linewidth=1)
plt.annotate(f'{min(Perr_m):4.2%}', fontsize=5,
                    xy=(threshold_best - .5, min(Perr_m)),
                    xytext=(threshold_best - 10, min(Perr_m)),
                    arrowprops=dict(arrowstyle='->'))
secondP.set_xlabel('threshold_m')
secondP.set_ylabel(r'$P_{error}$')
secondP.set_title('Estimate of the minimum probability of error')
plt.tight_layout()
plt.savefig('1_A5.png')
plt.show()

# ------ Part C ------
muhat0 = np.mean(true_x[:, ~true_data], axis=1)
sighat0 = np.cov(true_x[:, ~true_data])
muhat1 = np.mean(true_x[:, true_data], axis=1)
sighat1 = np.cov(true_x[:, true_data])

Scatter_2 = np.dot((muhat0 - muhat1).reshape(-1, 1), (muhat0 - muhat1).reshape(1, -1))

Sw = sighat0 + sighat1

lam, V = linalg.eig(Scatter_2, Sw)

# projection vector
wLDA = V[:, np.argmax(lam)]

divide_result_LDA = np.dot(wLDA.reshape(1, -1), true_x)

# choose the threshold value
divide_sort = np.sort(divide_result_LDA).ravel()
threshold_LDA = (divide_sort[0:-1] + divide_sort[1:]) / 2
```

```
threshold_LDA = np.insert(threshold_LDA, 0, divide_sort[0] - eps)
threshold_LDA = np.append(threshold_LDA, divide_sort[-1] + eps)

P_01_l = np.zeros(threshold_LDA.size)
P_10_l = np.zeros(threshold_LDA.size)
P_11_l = np.zeros(threshold_LDA.size)
Perr_l = np.zeros(threshold_LDA.size)
for i in range(threshold_LDA.size):
    LDA_decision = divide_result_LDA > threshold_LDA[i]
    P_01_l[i] = np.sum(~LDA_decision & true_data) / np.sum(true_data)
    P_10_l[i] = np.sum(LDA_decision & ~true_data) / np.sum(~true_data)
    P_11_l[i] = np.sum(LDA_decision & true_data) / np.sum(true_data)
    Perr_l[i] = P_01_l[i] * classifier[1] + P_10_l[i] * classifier[0]

# plot the ROC curve
figure3 = plt.figure(figsize=[6, 4], dpi=180)
ax1 = figure3.add_subplot(111)
ax1.plot(P_10_l, P_11_l, linewidth=1)
ax1.scatter(P_10_l[np.argmin(Perr_l)], P_11_l[np.argmin(Perr_l)], c='g',
            marker='x', label='minimum probability of error')
ax1.set_xlim([0, 1])
ax1.set_ylim([0, 1])
ax1.set_xlabel('Probability of false positive')
ax1.set_ylabel('Probability of true positive')
ax1.set_title('ROC curve by Fisher LDA classifer')
ax1.legend()
plt.tight_layout()
plt.savefig('1_A6.png')
plt.show()

# plot the probability of error
figure4 = plt.figure(figsize=[6, 4], dpi=180)
ax2 = figure4.add_subplot(111)
ax2.plot(threshold_LDA, Perr_l, linewidth=1)
plt.annotate(f'{min(Perr_l):4.2%}', fontsize=6,
             xy=(threshold_LDA[np.argmin(Perr_l)] - .1,
                 min(Perr_l)),
             xytext=(threshold_LDA[np.argmin(Perr_l)] - 2, min(Perr_l)),
             arrowprops=dict(arrowstyle='->'))
ax2.set_xlabel('threshold_LDA')
ax2.set_ylabel('probability pf error')
ax2.set_title('probability by Fisher LDA classifer')
plt.tight_layout()
plt.savefig('1_A7.png')
```

```
plt.show()

# make comparisons

figure5 = plt.figure(figsize=[6, 4], dpi=180)
ax3 = figure5.gca()
ax3.plot(P_10, P_11, linewidth=1, linestyle='-', c='r', label='ERM true pdf')
ax3.plot(P_10_m, P_11_m, linewidth=1, linestyle='--', c='y',
         label='ERM diagonal covariance pdf')
ax3.plot(P_10_l, P_11_l, linewidth=1, linestyle=':', c='b', label='LDA')
ax3.set_xlim([0, 1])
ax3.set_ylim([0, 1])
ax3.set_xlabel('Probability of false positive')
ax3.set_ylabel('Probability of true positive')
ax3.set_title('ROC curve comparison between three classifiers')
ax3.legend(fontsize=7)
plt.tight_layout()
plt.savefig('1_A8')
plt.show()

figure6 = plt.figure(figsize=[6, 4], dpi=180)
ax4 = figure6.gca()
ax4.plot(threshold_m, Perr_a, linewidth=1, linestyle='-', c='m',
         label='ERM pdf')
ax4.plot(threshold_m2, Perr_m, linewidth=1, linestyle='--', c='g',
         label='ERM diagonal covariance matrices pdf')
ax4.plot(threshold_LDA, Perr_l, linewidth=1, linestyle=':', c='b',
         label='fisher LDA')
ax4.set_xlabel('threshold')
ax4.set_ylabel('Probability of error')
ax4.set_title('Probability of error comparison between three classifiers')
ax4.legend(fontsize=8)
plt.tight_layout()
plt.savefig('1_A9')
plt.show()
```

Question2(Matlab)

```
clear all, close all,
C = 4;
N = 10000;
n = 3;
symbols='ox+*';
%Parameters for different classes
```

```
params.mean_scaling=4*(0:C-1);
params.Cov_scaling=3*(1:C) ;
gmmParameters.priors = [0.3 0.3 0.4];
gmmParameters.meanVectors(1,:) = params.mean_scaling;
gmmParameters.meanVectors(2,:) = params.mean_scaling;
gmmParameters.meanVectors(3,:) = params.mean_scaling;
%Define loss matrices
lossMatrix={'minErr' 'deltaU10' 'deltaU100'};
lossMatrixA.minErr = ones(C-1,C-1)-eye(C-1);
lossMatrixA.deltaU10= [0 1 10;
1 0 10;
1 1 0];
lossMatrixA.deltaU100= [0 1 100;
1 0 100;
1 1 0];
for ind = 1:C
gmmParameters.covMatrices(:,:,ind) =params.Cov_scaling(ind)*eye(n);
%A = params.Cov_scaling(ind)*eye(n);
%gmmParameters.covMatrices(:,:,ind) = A'*A; % arbitrary covariance matrices
end
% Generate data from specified pdf
[x,labels] = generateDataFromGMM(N,gmmParameters);
for ind = 1:3
Nclass(ind,1) = length(find(labels==ind));
end
C=C-1;
% Shared computation for both parts
for ind = 1:3
pxgivenl(ind,:)
=evalGaussianPDF(x,gmmParameters.meanVectors(:,ind),gmmParameters.covMatrices(:,:,in
d));
end
px = gmmParameters.priors*pxgivenl;
classPosteriors = pxgivenl.*repmat(gmmParameters.priors',1,N)./repmat(px,C,1);
%Plot Data with True Labels
figure;
for ind=1:C
plot3(x(1,labels==ind),x(2,labels==ind),x(3,labels==ind),symbols(ind),'DisplayName',['Class
' num2str(ind)]);
hold on;
end
xlabel('x1');
ylabel('x2');
grid on;
```

```
title('X Vector with True Data(True Classes)');
legend 'show';
%Classify Data based on loss values
for ind3=1:length(lossMatrix)
expectedRisksA.(lossMatrix{ind3})    =lossMatrixA.(lossMatrix{ind3})*classPosteriors;    %
Expected Risk for each label (rows) for each sample (columns)
[~,decisionsA.(lossMatrix{ind3})]       =min(expectedRisksA.(lossMatrix{ind3}),[],1);       %
Minimum expected risk decision with 0-1 loss is the same as MAP
fDecision_ind.(lossMatrix{ind3})=(decisionsA.(lossMatrix{ind3})~=labels);%Incorrect
classificiation vector
%Confusion Matrix
confMatrix.(lossMatrix{ind3})=zeros(C,C);
for ind=1:C
for ind2=1:C
confMatrix.(lossMatrix{ind3})(ind,ind2)=sum(decisionsA.(lossMatrix{ind3})==ind&
labels==ind2)/Nclass(ind2);
end
end

%Expected Risk
ExpRisk.(lossMatrix{ind3})=...
gmmParameters.priors*diag(expectedRisksA.(lossMatrix{ind3})'...
*confMatrix.(lossMatrix{ind3}));
fprintf('Expected Risk for %s=%1.2f\n',...
lossMatrix{ind3},ExpRisk.(lossMatrix{ind3}));
%Plot Decisions
figure;
for ind=1:C
class_ind=decisionsA.(lossMatrix{ind3})==ind;
plot3(x(1,class_ind),x(2,class_ind),x(3,class_ind),symbols(ind),...
'DisplayName',['Class ' num2str(ind)]);
hold on;
end

xlabel('x1');
ylabel('x2');
grid on;
title('X Vector with Classified (ERM) Values');
legend 'show';
%Plot Decisions with Incorrect Results as specified in assignment
figure;
for ind=1:C
class_ind=decisionsA.(lossMatrix{ind3})==ind;
plot3(x(1,class_ind & ~fDecision_ind.(lossMatrix{ind3})),...
```

```
x(2,class_ind & ~fDecision_ind.(lossMatrix{ind3})),...
x(3,class_ind & ~fDecision_ind.(lossMatrix{ind3})),...
symbols(ind),'Color',[0.39 0.83 0.07],'DisplayName',...
['Class ' num2str(ind) ' Correct Classification']);
hold on;
plot3(x(1,class_ind & fDecision_ind.(lossMatrix{ind3})),...
x(2,class_ind & fDecision_ind.(lossMatrix{ind3})),...
x(3,class_ind & fDecision_ind.(lossMatrix{ind3})),...
['r' symbols(ind)],'DisplayName',...
['Class ' num2str(ind) ' Incorrect Classification']);
hold on;
end
xlabel('x1');
ylabel('x2');
grid on;
title('X Vector with Correct and Incorrect Classifications');
legend 'show';
%Plot Decisions with Incorrect Decisions
figure;
for ind2=1:C
subplot(4,1,ind2);
for ind=1:C
class_ind=decisionsA.(lossMatrix{ind3})==ind;
plot3(x(1,class_ind),x(2,class_ind),x(3,class_ind),symbols(ind),'DisplayName',...
['Class ' num2str(ind)]);
hold on;
end

plot3(x(1,fDecision_ind.(lossMatrix{ind3}) & labels==ind2),...
x(2,fDecision_ind.(lossMatrix{ind3}) & labels==ind2),...
x(3,fDecision_ind.(lossMatrix{ind3}) & labels==ind2),...
'g.','DisplayName','Incorrectly Classified');
ylabel('x2');
grid on;
title(['X Vector with Incorrect Decisions for Class ' num2str(ind2)]);
if ind2==1
legend 'show';
elseif ind2==4
xlabel('x1');
end
end
end

function g = evalGaussian(x ,mu,Sigma)
```

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

function [x,labels] = generateDataFromGMM(N,gmmParameters)
% Generates N vector samples from the specified mixture of Gaussians
% Returns samples and their component labels
% Data dimensionality is determined by the size of mu/Sigma parameters
priors = gmmParameters.priors; % priors should be a row vector
meanVectors = gmmParameters.meanVectors;
covMatrices = gmmParameters.covMatrices;
n = size(gmmParameters.meanVectors,1); % Data dimensionality
C = length(priors); % Number of components
x = zeros(n,N); labels = zeros(1,N);
% Decide randomly which samples will come from each component
u = rand(1,N); thresholds = [cumsum(priors),1];
for l = 1:C
indl = find(u <= thresholds(l)); Nl = length(indl);
labels(1,indl) = l*ones(1,Nl);
u(1,indl) = 1.1*ones(1,Nl); % these samples should not be used again
x(:,indl) = mvnrnd(meanVectors(:,l),covMatrices(:,:,l),Nl)';
end
end

function px = evalGaussianPDF(x,mu,Sigma)
% x should have n-dimensional N vectors in columns
n = size(x,1); % data vectors have n-dimensions
N = size(x,2); % there are N vector-valued samples
C = (2*pi)^(-n/2)*det(Sigma)^(-1/2); % normalization constant
a = x-repmat(mu,1,N); b = inv(Sigma)*a;
% a,b are preparatory random variables, in an attempt to avoid a for loop
px = C*exp(-0.5*sum(a.*b,1)); % px is a row vector that contains p(x_i) values
end
```

Question3(Matlab)

```
clearvars; close all; clear all;

% Initial parameters
lambda=0.00001; %for regularization

% White Wine Quality Dataset
```

```
wine_raw_data = dlmread('winequality-white.csv',';',1,0)';

%Separate dataset into data and true class labels
x_W=wine_raw_data(1:end-1,:);
label_W=wine_raw_data(end,:);

% HAR Dataset (true class labels are separate files so no need to separate)
load X_train.txt;
load X_test.txt;
x_H=vertcat(X_train,X_test)';

%load true class labels
load Y_train.txt;
load Y_test.txt;
label_H=vertcat(Y_train, Y_test)';

% Dimensions and size of datasets from matrices
[n_W, N_W]=size(x_W);
[n_H,N_H]=size(x_H);

% Class labels and number of classes
class_W=unique(label_W);
C_W=length(class_W);
class_H=unique(label_H);
C_H=length(class_H);

% Class priors
priors_W=zeros(1,C_W);
for l=1:C_W
	priors_W(l)=sum(label_W==class_W(l))/N_W;
end
priors_H=zeros(1,C_H);
for l=1:C_H
	priors_H(l)=sum(label_H == class_H(l))/N_H;
end

% Estimated mean vectors
mu_W=zeros(n_W, C_W);
for l=1:C_W
	samples=x_W(:,label_W == class_W(l));
	mu_W(:,l)=mean(samples, 2);
end
mu_H=zeros(n_H, C_H);
for l=1:C_H
```

```
    samples=x_H(:,label_H == class_H(l));
    mu_H(:,l)=mean(samples, 2);
end

% Estimated covariance matrices
Sigma_W=zeros(n_W, n_W, C_W);
for l=1:C_W
    Sigma_W(:,:,l)=cov(x_W(:,label_W==class_W(l))')+(lambda*eye(n_W));
end
Sigma_H = zeros(n_H, n_H, C_H);
for l=1:C_H
    Sigma_H(:,:,l)=cov(x_H(:,label_H==class_H(l))')+(lambda*eye(n_H));
end

%[QW,DW]=eig(Sigma_W(:,:,1));
%[QH,DH]=eig(Sigma_H(:,:,1));

% Class posteriors and loss matrices
classPosterior_W=classPosterior(x_W, mu_W, Sigma_W, N_W, C_W, priors_W);
classPosterior_H=classPosterior_Mvnpdf(x_H, mu_H, N_H, C_H, priors_H);

lossMatrix_W=zeros(C_W);
 for i=1:C_W
    for j=1:C_W
        lossMatrix_W(i,j) = abs(i-j);
    end
 end
lossMatrix_H=ones(C_H,C_H)-eye(C_H);

% Run classification for confusion matrices and create pError
[decisions_W,confusionMatrix_W]=runClassif(lossMatrix_W, classPosterior_W, label_W,
class_W, priors_W);
[decisions_H,confusionMatrix_H]=runClassif(lossMatrix_H, classPosterior_H, label_H,
class_H, priors_H);
pError_W=calculatePErr(confusionMatrix_W, priors_W);
pError_H=calculatePErr(confusionMatrix_H, priors_H);
y_W=LDA(x_W',label_W)';
y_H=LDA(x_H',label_H)';

%Plot first 3 dimensions of LDAprojections on each dataset
figure(1);
for l=1:C_W
    scatter3(y_W(1, label_W==class_W(l)), y_W(2, label_W==class_W(l)), y_W(3,
label_W==class_W(l)), '.');
```

```
    hold on;
    axis equal;
end
title('White Wine Dataset (using Fisher LDA) - 3 Dimensions');
xlabel('y1'); ylabel('y2'); zlabel('y3');
% legend('Class 3', 'Class 4', 'Class 5', 'Class 6', 'Class 7', 'Class 8', 'Class 9');

legend('Class 1', 'Class 2', 'Class 3', 'Class 4', 'Class 5', 'Class 6', 'Class 7');

figure(2);
for l=1:C_H
    scatter3(y_H(1, label_H==class_H(l)), y_H(2, label_H==class_H(l)), y_H(3,
label_H==class_H(l)), '.');
    hold on;
    axis equal;
end
title('HAR Dataset (using Fisher LDA) - 3 Dimensions');
xlabel('y1'); ylabel('y2'); zlabel('y3');
% legend('Walking', 'Walking Upstairs', 'Walking Downstairs', 'Sitting', 'Standing', 'Laying')

legend('Class 1', 'Class 2', 'Class 3', 'Class 4', 'Class 5', 'Class 6')


%given a confusion matrix and corresponding class priors calculate
%probability of error for the classifier
function pErr=calculatePErr(confusionMatrix, prior)
    C=length(prior);
    pErr=0;
    for l=1:C
        for d=1:C
            if d~=l
                pErr=pErr+confusionMatrix(d,l) * prior(l);
            end
        end
    end
end

% Class posterior for samples
function p=classPosterior(x, mu, Sigma, N, C, priors)
    for l=1:C
        pxgivenl(l,:)=evalGaussian(x,mu(:,l),Sigma(:,:,l)');
    end
    px=priors*pxgivenl;
    p=pxgivenl.*repmat(priors',1,N)./repmat(px,C,1);
```

```
end

% Class posterior for HAR dataset using mvnpdf and omitting sigma, since the large covariance
% matrices cause issues
function p=classPosterior_Mvnpdf(x, mu, N, C, priors)
    for l=1:C
        pxgivenl(l,:)=mvnpdf(x',mu(:,l)');
    end
    px=priors*pxgivenl;
    p=pxgivenl.*repmat(priors',1,N)./repmat(px,C,1);
end

% Make decisions and confusion matrix
function[decision, confusionMatrix]=runClassif(lossMatrix, classPosterior, label, class,
labelCount)
    C=length(class);
    expRisk=lossMatrix*classPosterior;
    [~,decisionInds]=min(expRisk,[],1);
    decision=class(decisionInds);

    confusionMatrix=zeros(C);
    for l=1:C
        classDecision=decision(label==class(l));
        for d=1:C
            confusionMatrix(d,l)=sum(classDecision==d)/labelCount(l);
        end
    end
end

% evalGaussian
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

% LDA
function Y = LDA(X,L)
    Classes=unique(L)';
    k=numel(Classes);
    n=zeros(k,1);
    C=cell(k,1);
```

```
    M=mean(X);
    S=cell(k,1);
    Sw=0;
    Sb=0;
    for j=1:k
        Xj=X(L==Classes(j),:);
        n(j)=size(Xj,1);
        C{j}=mean(Xj);
        S{j}=0;
        for i=1:n(j)
            S{j}=S{j}+(Xj(i,:)-(C{j})'*(Xj(i,:)-C{j}));
        end
        Sw=Sw+S{j};
        Sb=Sb+n(j)*(C{j}-M)'*(C{j}-M);
    end
    [W, LAMBDA]=eig(Sb,Sw);
    lambda=diag(LAMBDA);
    [~, SortOrder]=sort(lambda,'descend');
    W=W(:,SortOrder);
    Y=X*W;
end
```