```python
# Import Necessary Modules
import glob
import matplotlib.pyplot as plt
from skimage import io
import numpy as np
import math
import scipy.stats as stats
%matplotlib inline

#This function takes in the column vectors of guesses for H0 and H1
#data and outputs the probability of error, assuming H0 and H1 are
#equally likely.

def proberror(testguess0,testguess1):
    #Calculate sizes.
    n0test = testguess0.shape[0]
    n1test = testguess1.shape[0]
    #The number of false alarms is the total number of ones in
testguess0.
    num_false_alarm =
np.sum(np.not_equal(testguess0,np.zeros((n0test,1))))
    #Divide by total number of guesses to estimate the probability of
false alarm.
    P_FA = num_false_alarm / n0test
    #The number of missed detections is the total number of zeros in
testguess1.
    num_missed_detection =
np.sum(np.not_equal(testguess1,np.ones((n1test,1))))
    #Divide by total number of guesses to estimate the probability of
missed detection.
    P_MD = num_missed_detection / n1test
    Pe = P_FA * 0.5 + P_MD * 0.5
    return Pe

#Read in synthetic data for 8.5(a), 8.5(b), 8.5(c)
dataset0 = np.genfromtxt("syntheticH0.csv", delimiter = ",")
dataset1 = np.genfromtxt("syntheticH1.csv", delimiter = ",")

#Read in pet classificaton data for 8.4(h)
# dataset0,dataset1 = read_cats_dogs()

#Determine number of samples and dimension.
n0,d0 = dataset0.shape
n1,d1 = dataset1.shape
if (d0 == d1):
    d = d0
else:
    raise Exception("dataset0 and dataset1 do not have the same number
of columns.")
```

```
#Split dataset into training and test data.
train0 = dataset0[0:math.floor(n0/2),:]
test0 = dataset0[math.floor(n0/2):n0,:]
train1 = dataset1[0:math.floor(n1/2),:]
test1 = dataset1[math.floor(n1/2):n1,:]
n0train = train0.shape[0]
n1train = train1.shape[0]
n0test = test0.shape[0]
n1test = test1.shape[0]

#Estimate mean vectors and covariance matrices from training data.
mu0 = np.mean(train0, axis=0)
mu1 = np.mean(train1, axis=0)
sigma0 = np.cov(train0, rowvar=False)
sigma1 = np.cov(train1, rowvar=False)

#This function takes in a row vector currentdata as well as two mean
#vectors mu0 and m1. It outputs 1 if currentdata is closer to mu1
#than mu0, and 0 if mu0 is closer. In the case of a tie, it outputs 1.

def closest_average(currentdata,mu0,mu1):
    #Calculate distances.
    distance_mu0 = np.math.sqrt(np.sum((currentdata-mu0)**2))
    distance_mu1 = np.math.sqrt(np.sum((currentdata-mu1)**2))
    #Decide based on smaller distance.
    if (distance_mu0 < distance_mu1):
        guess = 0
    elif (distance_mu0 >= distance_mu1):
        guess = 1
    return guess

#8.5(a) Apply Gaussian ML rule for identity covariance matrix by
plugging into the PDFs.
H0guesses_idcov = np.zeros((n0test,1))
H1guesses_idcov = np.zeros((n1test,1))

for i in range(n0test):
    currentdata = test0[i,:]
    h0 = stats.multivariate_normal.pdf(currentdata, mu0,
np.identity(d0))
    h1 = stats.multivariate_normal.pdf(currentdata, mu1,
np.identity(d0))
    if h1>= h0:
        H0guesses_idcov[i] =  1

for i in range(n1test):
    currentdata = test1[i,:]
    h0 = stats.multivariate_normal.pdf(currentdata, mu0,
np.identity(d1))
    h1 = stats.multivariate_normal.pdf(currentdata, mu1,
```

```
np.identity(d1))
    if h1>= h0:
        H1guesses_idcov[i] = 1

Pe_idcov = proberror(H0guesses_idcov,H1guesses_idcov)
print("Probability of error for identity covariance matrix is " +
str(Pe_idcov) + ".")
```

```
Probability of error for identity covariance matrix is 0.068.
```

```
#8.5 (b) Apply Gaussian ML rule for the same covariance matrix by
plugging into the PDFs.
H0guesses_samecov = np.zeros((n0test,1))
H1guesses_samecov = np.zeros((n1test,1))

sigmas = (1/(n0train+n1train-2))*((n0train-1)*sigma0+(n1train-
1)*sigma1)

for i in range(n0test):
    currentdata = test0[i,:]
    h0 = stats.multivariate_normal.pdf(currentdata, mu0, sigmas)
    h1 = stats.multivariate_normal.pdf(currentdata, mu1, sigmas)
    if h1>=h0:
        H0guesses_samecov[i] = 1

for i in range(n1test):
    currentdata = test1[i,:]
    h0 = stats.multivariate_normal.pdf(currentdata, mu0, sigmas)
    h1 = stats.multivariate_normal.pdf(currentdata, mu1, sigmas)
    if h1>=h0:
        H1guesses_samecov[i] = 1

Pe_samecov = proberror(H0guesses_samecov,H1guesses_samecov)
print("Probability of error for the same covariance matrices is " +
str(Pe_samecov) + ".")
```

```
Probability of error for the same covariance matrices is 0.05.
```

```
#8.5 c Apply Gaussian ML rule for different covariance matrices by
plugging into the PDFs.
H0guesses_diffcov = np.zeros((n0test,1))
H1guesses_diffcov = np.zeros((n1test,1))

for i in range(n0test):
    currentdata = test0[i,:]
    h0 = stats.multivariate_normal.pdf(currentdata, mu0, sigma0)
    h1 = stats.multivariate_normal.pdf(currentdata, mu1, sigma1)
    if h1>=h0:
        H0guesses_samecov[i] = 1

for i in range(n1test):
```

```
    currentdata = test1[i,:]
    h0 = stats.multivariate_normal.pdf(currentdata, mu0, sigma0)
    h1 = stats.multivariate_normal.pdf(currentdata, mu1, sigma1)
    if h1>=h0:
        H1guesses_diffcov[i] = 1

Pe_diffcov = proberror(H0guesses_diffcov,H1guesses_diffcov)
print("Probability of error for different covariance matrices is " +
str(Pe_diffcov) + ".")
```

```
Probability of error for different covariance matrices is 0.012.
```

```
## For problem 8.5(d), load a new data set:

# Read in breast cancer data
dataset0 = np.genfromtxt("benignfull.csv", delimiter = ",")
dataset1 = np.genfromtxt("malignantfull.csv", delimiter = ",")

# compute the statistics
n0,d0 = dataset0.shape
n1,d1 = dataset1.shape
if (d0 == d1):
    d = d0
else:
    raise Exception("dataset0 and dataset1 do not have the same number
of columns.")

#Split dataset into training and test data.
train0 = dataset0[0:math.floor(n0/2),:]
test0 = dataset0[math.floor(n0/2):n0,:]
train1 = dataset1[0:math.floor(n1/2),:]
test1 = dataset1[math.floor(n1/2):n1,:]
n0train = train0.shape[0]
n1train = train1.shape[0]
n0test = test0.shape[0]
n1test = test1.shape[0]

#Estimate mean vectors and covariance matrices from training data.
mu0 = np.mean(train0, axis=0)
mu1 = np.mean(train1, axis=0)
sigma0 = np.cov(train0, rowvar=False)
sigma1 = np.cov(train1, rowvar=False)

#8.5(d) Apply Gaussian ML rule for identity covariance matrix by
plugging into the PDFs for the cancer data
#  Note: you shoud reuse the code from 8.5(a)
H0guesses_idcov = np.zeros((n0test,1))
H1guesses_idcov = np.zeros((n1test,1))

for i in range(n0test):
    currentdata = test0[i,:]
```

```
    h0 = stats.multivariate_normal.pdf(currentdata, mu0,
np.identity(d0))
    h1 = stats.multivariate_normal.pdf(currentdata, mu1,
np.identity(d0))
    if h1>=h0:
        H0guesses_idcov[i] = 1

for i in range(n1test):
    currentdata = test1[i,:]
    h0 = stats.multivariate_normal.pdf(currentdata, mu0,
np.identity(d1))
    h1 = stats.multivariate_normal.pdf(currentdata, mu1,
np.identity(d1))
    if h1>=h0:
        H1guesses_idcov[i] = 1

Pe_idcov = proberror(H0guesses_idcov,H1guesses_idcov)
print("Probability of error for identity covariance matrix is " +
str(Pe_idcov) + ".")
```

```
Probability of error for identity covariance matrix is 0.385.
```

```
#Now we start testing with the log-likelihood tests
# Problem 8.5(e)  Apply Gaussian ML rule for identity covariance
matrix by using the closest average classifier.
# Recall that function was defined above.

H0guesses_idcov = np.zeros((n0test,1))
H1guesses_idcov = np.zeros((n1test,1))

for i in range(n0test):
    currentdata = test0[i,:]
    H0guesses_idcov[i] = closest_average(currentdata, mu0, mu1)

for i in range(n1test):
    currentdata = test1[i,:]
    H1guesses_idcov[i] = closest_average(currentdata,mu0,mu1)

Pe_idcov = proberror(H0guesses_idcov,H1guesses_idcov)
print("Probability of error for identity covariance matrix is " +
str(Pe_idcov) + ".")
```

```
Probability of error for identity covariance matrix is
0.08499999999999999.
```

```
/var/folders/pf/619m_yfj3w5cwc52r3yxbb_40000gn/T/
ipykernel_46189/403505849.py:7: DeprecationWarning: `np.math` is a
deprecated alias for the standard library `math` module (Deprecated
Numpy 1.25). Replace usages of `np.math` with `math`
  distance_mu0 = np.math.sqrt(np.sum((currentdata-mu0)**2))
```

```
/var/folders/pf/619m_yfj3w5cwc52r3yxbb_40000gn/T/ipykernel_46189/40350
5849.py:8: DeprecationWarning: `np.math` is a deprecated alias for the
standard library `math` module (Deprecated Numpy 1.25). Replace usages
of `np.math` with `math`
  distance_mu1 = np.math.sqrt(np.sum((currentdata-mu1)**2))
```

```
#8.5 (f) Apply Gaussian ML rule for the same covariance matrix by
using the log-likelihood ratio
# using poled covariances
H0guesses_samecov = np.zeros((n0test,1))
H1guesses_samecov = np.zeros((n1test,1))

sigmas = (1/(n0train+n1train-2))*((n0train-1)*sigma0+(n1train-
1)*sigma1)
sigmainv = np.linalg.pinv(sigmas)

for i in range(n0test):
    currentdata = test0[i,:]
    value0 = np.matmul(np.matmul((currentdata-
mu0),sigmainv),np.transpose(currentdata-mu0))
    value1 = np.matmul(np.matmul((currentdata-
mu1),sigmainv),np.transpose(currentdata-mu1))
    if value1<=value0:
        H0guesses_samecov[i] = 1

for i in range(n1test):
    currentdata = test1[i,:]
    value0 = np.matmul(np.matmul((currentdata-
mu0),sigmainv),np.transpose(currentdata-mu0))
    value1 = np.matmul(np.matmul((currentdata-
mu1),sigmainv),np.transpose(currentdata-mu1))
    if value1<=value0:
        H1guesses_samecov[i] = 1

Pe_samecov = proberror(H0guesses_samecov,H1guesses_samecov)
print("Probability of error for the same covariance matrices is " +
str(Pe_samecov) + ".")
```

```
Probability of error for the same covariance matrices is 0.065.
```

```
#8.5(g) Apply Gaussian ML rule for different covariance matrices by
using the log-likelihood ratio.
H0guesses_diffcov = np.zeros((n0test,1))
H1guesses_diffcov = np.zeros((n1test,1))

sigma0inv = np.linalg.pinv(sigma0)
sigma1inv = np.linalg.pinv(sigma1)
offset = math.log(np.linalg.det(sigma0))-
math.log(np.linalg.det(sigma1))
```

```
for i in range(n0test):
    currentdata = test0[i,:]
    value0 = np.matmul(np.matmul((currentdata-
mu0),sigma0inv),np.transpose(currentdata-mu0))
    value1 = np.matmul(np.matmul((currentdata-
mu1),sigma1inv),np.transpose(currentdata-mu1))
    if value1<=(value0+offset):
        H0guesses_diffcov[i] = 1

for i in range(n1test):
    currentdata = test1[i,:]
    value0 = np.matmul(np.matmul((currentdata-
mu0),sigma0inv),np.transpose(currentdata-mu0))
    value1 = np.matmul(np.matmul((currentdata-
mu1),sigma1inv),np.transpose(currentdata-mu1))
    if value1<=(value0+offset):
        H1guesses_diffcov[i] = 1

Pe_diffcov = proberror(H0guesses_diffcov,H1guesses_diffcov)
print("Probability of error for different covariance matrices is " +
str(Pe_diffcov) + ".")
```

```
Probability of error for different covariance matrices is 0.015.
```