Roll No.: CB.EN.U4CSE19453

Amrita Vishwa Vidyapeetham

Amrita School of Engineering, Coimbatore

B.Tech Internal Assessment – September 2022

Seventh Semester

Computer Science Engineering

19CSE432 Pattern Recognition

Maximum: 5 Marks

Course Outcomes (COs):

CO	Course Outcomes
CO01	Understand basic concepts in pattern recognition
CO02	Understand discriminant functions and apply them for applications..
CO03	Understand and apply Parametric techniques of Pattern recognition..
CO04	Apply Non parametric techniques of PR and analyze their performance..
CO05	Understand the supervised and unsupervised learning algorithms and apply them for real world problems.

2.2 :

1)

A) Write a procedure to generate random samples according to a normal distribution N(μ, Σ) in d dimensions.

Solution:

To generate five random numbers from the normal distribution we will use numpy.random.normal() method of the random module

Code:

```
import numpy as np
import math
r = np.random.normal(0,1,(3,3))
print(r)
```

```
[[ 0.40134634 -1.35248569 -0.886
 [ 0.84111147  1.40428928 -0.106
```

Explanation:

Here, In the code: 0 is the mean of distribution,
1 is the standard deviation,
And (3,3) is the shape for the resultant

B) Write a procedure to calculate the discriminant function (of the form given in Eq. 47) for a given normal distribution and prior probability $P(\omega_i)$.

Formula:

$$g_i(x) = -\frac{1}{2}(x-\mu_i)^t\Sigma_i^{-1}(x-\mu_i) - \frac{d}{2}\ln 2\pi - \frac{1}{2}\ln|\Sigma$$

Code:

```
import numpy as np
import math

def discriminant(x,mu,sigma,prior):
    x = np.array(x)
    mu = np.array(mu)
    sigma = np.array(sigma)
    x_mu = x - mu
    inv_sigma = np.linalg.inv(sigma)
    return -0.5*((x_mu))*(inv_sigma)*(x_mu) -
(0.5*len(x)*math.log(2*math.pi))-
(0.5*math.log(np.linalg.det(sigma)))+math.log(prior)

def main():
    x = [1,2,3]
    mu = [0,0,0]
    sigma = [[1,0,0],[0,1,0],[0,0,1]]
    prior = 1
    print(discriminant(x,mu,sigma,prior))

main()
```

Output:

```
[[-3.2568156 -2.7568156 -2.75
 [-2.7568156 -4.7568156 -2.75
 [-2.7568156 -2.7568156 -7.25
```

C) Write a procedure to calculate the Euclidean distance between two arbitrary points.

Formula:

$$d(p,q) = \sqrt[2]{(q_1 - p_1)^2 + (q_2 - p_2)^2}$$

Code:

```
import numpy as np

point1 = np.array((3, 2))
point2 = np.array((4, 1))

sum_sq = np.sum(np.square(point1 - point2))

print(np.sqrt(sum_sq))
def euclidean(x,y):
    x = np.array(x)
    y = np.array(y)
    return math.sqrt(np.dot(np.dot(x, x), y, y))
def main():
    x = [3,2]
    y = [4,1]
    print(euclidean(x,y))
```

Output:

```
1.4142135623730951
```

D) Write a procedure to calculate the Mahalanobis distance between the mean μ and an arbitrary point x, given the covariance matrix Σ.

Formula:

- $\mathbf{D^2 = (x\text{-}m)^T C^{-1} (x\text{-}m)}$

Code:

```
import numpy as np
import math

# Mahalanobis distance
def mahalanobis(x=None, data=None, cov=None):
    x_mu = x - np.mean(data)
    if not cov:
        cov = np.cov(data.values.T)
    inv_covmat = np.linalg.inv(cov)
    left = np.dot(x_mu, inv_covmat)
    mahal_dist = np.dot(left, x_mu.T)
    return mahal_dist.diagonal()
```

2.5:

sample	ω_1 x_1	x_2	x_3	ω_2 x_1	x_2	x_3	x_1
1	−5.01	−8.12	−3.68	−0.91	−0.18	−0.05	5.35
2	−5.43	−3.48	−3.54	1.30	−2.06	−3.53	5.12
3	1.08	−5.52	1.66	−7.75	−4.54	−0.95	−1.34
4	0.86	−3.78	−4.11	−5.47	0.50	3.92	4.48
5	−2.67	0.63	7.39	6.14	5.72	−4.85	7.11
6	4.94	3.29	2.08	3.60	1.26	4.36	7.17
7	−2.51	2.09	−2.59	5.37	−4.63	−3.65	5.75
8	−2.25	−2.13	−6.94	7.18	1.46	−6.66	0.77

2)

A) Assume that the prior probabilities for the first two categories are equal (P(ω1) = P(ω2)=1/2 and P(ω3) = 0) and design a dichotomizer for those two categories using only the x1 feature value.

D) Repeat all of the above, but now use two feature values, x1, and x2.

E) Repeat, but use all three feature values.

Implementation:

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as st
import math


def bhattacharyya(prior1, prior2, mean1, mean2, cov1, cov2):
    return 0.125 * np.log(np.linalg.det(np.linalg.inv(0.5 * cov1 + 0.5 * cov2))) +
0.5 * np.dot(np.dot((mean1 - mean2), np.linalg.inv(0.5 * cov1 + 0.5 * cov2)),
(mean1 - mean2).T)


def classifier1(x, class1, class2):
    prior1 = 0.5
    prior2 = 0.5
    mean1 = np.mean(class1[:, 0:2], axis=0)
    mean2 = np.mean(class2[:, 0:2], axis=0)

    # print mean1, mean2
    cov1 = np.cov([class1[:, 0], class1[:, 1]])
    cov2 = np.cov([class2[:, 0], class2[:, 1]])

    # print cov1, cov2
    discriminant_function1 = gdf.gen_discriminant_function_of_normal_distribution(
        mean1, cov1, prior1)
    discriminant_function2 = gdf.gen_discriminant_function_of_normal_distribution(
        mean2, cov2, prior2)

    if discriminant_function1(x) > discriminant_function2(x):
```

```python
        # print x, "class 1"
        return 1, bhattacharyya(prior1, prior2, mean1, mean2, cov1, cov2)
    elif discriminant_function1(x) < discriminant_function2(x):
        # print x, "class 2"
        return 2, bhattacharyya(prior1, prior2, mean1, mean2, cov1, cov2)
    else:
        # print x, "unsure"
        return 0, bhattacharyya(prior1, prior2, mean1, mean2, cov1, cov2)
```

3) Repeat Computer exercise 2 but for categories ω1 and ω3.

4) Repeat Computer exercise 2 but for categories ω2 and ω3.

5) Consider the three categories in Computer exercise 2, and assume $P(\omega_i)=1/3$.

(a) What is the Mahalanobis distance between each of the following test points and each of the category means in Computer exercise 2: $(1, 2, 1)^t$, $(5, 3, 2)^t$, $(0, 0, 0)^t$, $(1, 0, 0)^t$.

(b) Classify those points.

(c) Assume instead that P(ω1)=0.8, and P(ω2) = P(ω3)=0.1 and classify the test points again.

Solution:

Here from the question let,

X_1 = [[-5.01,-5.43,1.08,0.86,-2.67,4.94,-2.51,-2.25,5.56,1.03]',

[-8.12,-3.48,-5.52,-3.78,0.63,3.29,2.09,-2.13,2.86,-3.33]',

[-3.68,-3.54,1.66,-4.11,7.39,2.08,-2.59,-6.94,-2.26,4.33]']

X_2= [[-0.91,1.30,-7.75,-5.47,6.14,3.60,5.37,7.18,-7.39,-7.50]',

[-0.18,-2.06,-4.54,0.50,5.72,1.26,-4.63,1.46,1.17,-6.32]',

[-0.05,-3.53,-0.95,3.92,-4.85,4.36,-3.65,-6.66,6.30,-0.31]'];

X_3 = [[5.35,5.12,-1.34,4.48,7.11,7.17,5.75,0.77,0.90,3.52]',

[2.26,3.22,-5.31,3.42,2.39,4.33,3.97,0.27,-0.43,-0.36]',

[8.13,-2.66,-9.87,5.19,9.21,-0.98,6.65,2.41,-8.71,6.43]]

And prior probabilities, mean, covariances are:

- p1 = [1,2,1]';
- p2 = [5,3,1]';
- p3 = [0,0,0]';
- p4 = [1,0,0]';

prior1 = 0.8	mu1 = (mean(x1))'	sigma1 = cov(x1)
prior2 = 0.1	mu2 = (mean(x2))'	sigma2 = cov(x2)
prior3 = 0.1	mu3 = (mean(x3))';	sigma3 = cov(x3);

- g11 = gaussiandiscriminant(p1,mu1,sigma1,prior1)
- g12 = gaussiandiscriminant(p1,mu2,sigma2,prior2)
- g13 = gaussiandiscriminant(p1,mu3,sigma3,prior3)
- [g1,d1]=max([g11,g12,g13])

- g21 = gaussiandiscriminant(p2,mu1,sigma1,prior1)
- g22 = gaussiandiscriminant(p2,mu2,sigma2,prior2)
- g23 = gaussiandiscriminant(p2,mu3,sigma3,prior3)
- [g2,d2]=max([g21,g22,g23])

- g31 = gaussiandiscriminant(p3,mu1,sigma1,prior1)
- g32 = gaussiandiscriminant(p3,mu2,sigma2,prior2)
- g33 = gaussiandiscriminant(p3,mu3,sigma3,prior3)
- [g3,d3]=max([g31,g32,g33])

- g41 = gaussiandiscriminant(p4,mu1,sigma1,prior1)
- g42 = gaussiandiscriminant(p4,mu2,sigma2,prior2)
- g43 = gaussiandiscriminant(p4,mu3,sigma3,prior3)
- [g4,d4]=max([g41,g42,g43])

➢ d = [d1,d2,d3,d4]'

Now by using code below we get discriminant function for four data points:

```
[n, m] = size(samples)
for i in range(1, 3):
    mu[i] = mean(samples(:, (i-1)*3+1: i*3))
    sigma[i] = zeroes(3)
    for j in range(1, n):
        sigma[i] = sigma[i] + (samples(j, (i-1)*3+1: i*3) - mu[i])*(samples(j, (i-1)*3+1: i*3) -
mu[i])
    end
    sigma[i] = sigma[i]/n
```

```
end

s = [1 2 1
     5 3 2
     0 0 0
     1 0 0]'

for j in range(1, size(s, 2)):
    for i in range(1, 3):
        d = sqrt((s(: , j) - mu(i))'* inv(sigma(i))*(s(: , j) - mu(i)))
        print('Mahabolanobis distance for sample %d and class %d is %f' % (j, i, d))
    end
end

pw(1, :) = [1/3 0.8]
pw(2, :) = [1/3 0.1]
pw(3, :) = [1/3 0.1]

for p in range(1, 2):
    print('\n\n\n\n')
    for j in range(1, size(s, 2)):
        cla = 0
        max_gi = -1000000
        for i in range(1, 3):
            d = (s(: , j) - mu(i))'* inv(sigma(i))*(s(: , j) - mu(i))
            gi = -0.5*d - 1.5*log(2*pi) - 0.5 * \
                log(det(sigma(i))) + log(pw(i, p))
            if gi > max_gi:
                max_gi = gi
                cla = i
            end
        end
        print('Sample %d belongs to class %d' % (j, cla))
    end
end
```

$g_i(x_j)$	G1	G2	G3
X1	-7.45	-9.49	-11.6
X2	-8.13	-10.28	-8.3
X3	-7.06	-9.16	-10.5
X4	-7.05	-9.23	-9.14

Applying rule of maximum discriminant function, the classification results are

$$d1=d2=d3=d4=1$$

i.e., all 4 data points lie in first category

Mahalanobis distances between the three points and mean vector using above code is:

(X_i, μ_j)	μ_1	μ_2	μ_3
X_1	1.01	0.85	2.67
X_2	1.54	1.51	0.74

X_3	0.49	0.26	2.24
X_4	0.48	0.45	1.46

Now applying rule of mahalanobis distance, the classification is:

d1=2, d2=3, d3=2, d4=2

6. Illustrate the fact that the average of a large number of independent random variables will approximate a Gaussian by the following:

(a) Write a program to generate n random integers from a uniform distribution U(xl, xu). (Some computer systems include this as a single, compiled function call.)

Formula:

- v = a + (b-a).*rand(N,1)

Code:

```
def uniform_distribution(xl, xu, n):
    # x = np.random.uniform(xl, xu, n)
    x = (xu-xl)*np.random.rand(n) + xl
    return x

def main():
    xl = 0
    xu = 1
    n = 10
    print(uniform_distribution(xl, xu, n))
main()
```

Output:

```
[0.80488261 0.85396388 0.29878133 0.14210106 0.3626601
 0.07332778 0.88406115 0.86685774 0.38777871]
```

(b) Now write a routine to choose x_1 and x randomly, in the range $-100 \leq xl < xu \leq +100$, and n (the number of samples) randomly in the range $0 < n \leq 1000$.

Code:

```
import numpy as np
import random
```

```python
def uniform_distribution(xl, xu, n):
    for xl in range(-100,100):

        if xl <= xu:
            x = np.random.uniform(xl, xu, n)
            return x
        else:
            print('xl must be less than xu')

def main():
    xl = random.randrange(-100, 100)
    print("xl=",xl)
    xu = random.randrange(xl, 100)
    print("xu=",xu)
    n = random.randrange(0, 1000)
    print("n=",n)
    print(uniform_distribution(xl, xu, n))
main()
```

Output:

```
xl= 25
n= 272
xu= 53
[-81.65394176 -63.00759522  30.55584124  34.3713052  -77
 -52.44995052  34.12619047 -51.30382895   3.28824798  -1
  24.528094    27.08268068 -68.52846202 -59.21231672 -42
 -56.8833968  -23.71115851 -87.61501328 -68.0991025   32
  51.08392357  10.42960652  -0.27239704  26.96888395 -90
  13.87322374 -65.31867868 -62.00337716 -58.35241338 -17
 -78.00840439  10.02837633 -76.50499556  17.8605739   47
 -55.24957994 -39.66959578  19.92981693 -35.53800961 -97
 -58.27953289 -21.71322401 -75.62805768 -80.30941616   1
 -96.48283117  -0.93962738   8.75421628 -72.85169392  44
 -83.65433015   6.97796447 -13.61687936 -31.73618706  46
```

(c) Generate and plot a histogram of the accumulation of 10^4 points sampled as just described.

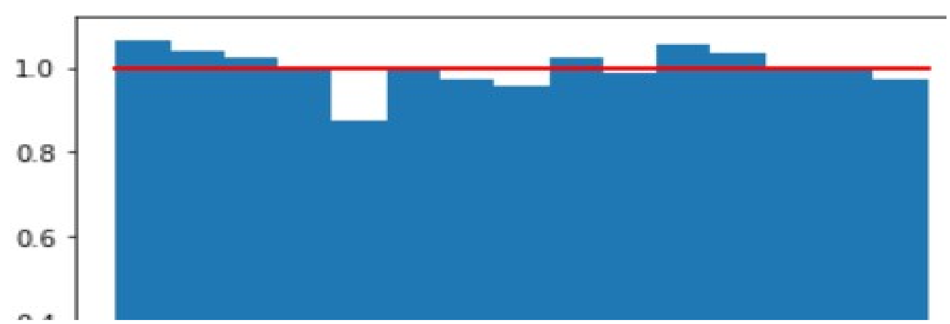
Code:

```
[20] import numpy as np
     s = np.random.uniform(-1,0,10000)
```

```
[21] print(np.all(s >= -1))
     print(np.all(s < 0))
```

```
True
True
```

```
import matplotlib.pyplot as plt
count, bins, ignored = plt.hist(s, 15, density
plt.plot(bins, np.ones_like(bins), linewidth=2
plt.show()
```



(d) Calculate the mean and standard deviation of your histogram, and plot it

Code:

```
print("Mean of samples genearated:",np.m
print("Standard Deviation of samples:",n
```

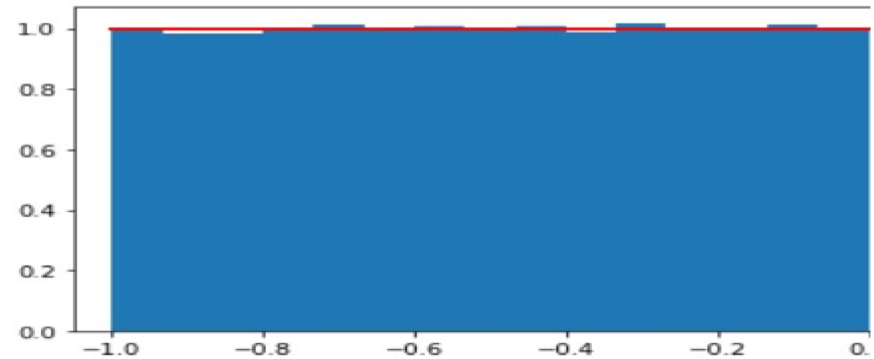
```
Mean of samples genearated: -0.501635225
```

(e) Repeat the above for 10^5 and for 10^6. Discuss your results.

Code:

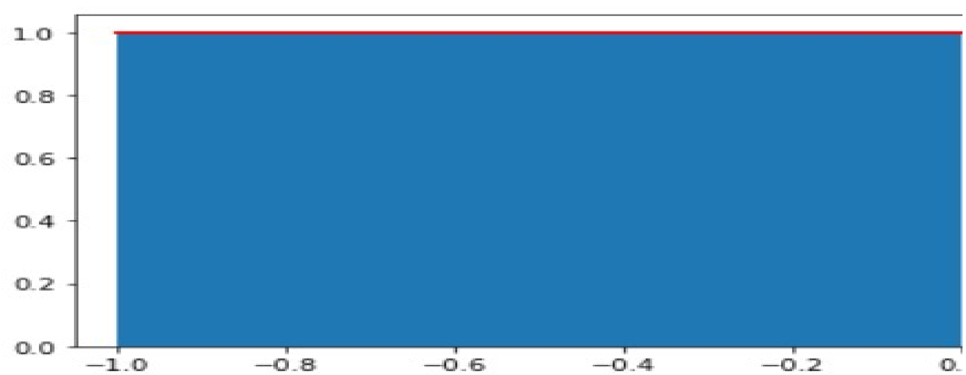
For 10^5:

```
[29] import matplotlib.pyplot as plt
     import numpy as np
     s = np.random.uniform(-1,0,100000)
     count, bins, ignored = plt.hist(s, 15, densit
     plt.plot(bins, np.ones_like(bins), linewidth=
     plt.show()
```



For 10^6:

```
[31] import matplotlib.pyplot as plt
     import numpy as np
     s = np.random.uniform(-1,0,1000000)
     count, bins, ignored = plt.hist(s, 15, densit
     plt.plot(bins, np.ones_like(bins), linewidth=
     plt.show()
```



Inference:

As the value of n increases, the mean remains same close to -0.5 and std close to 0.28.

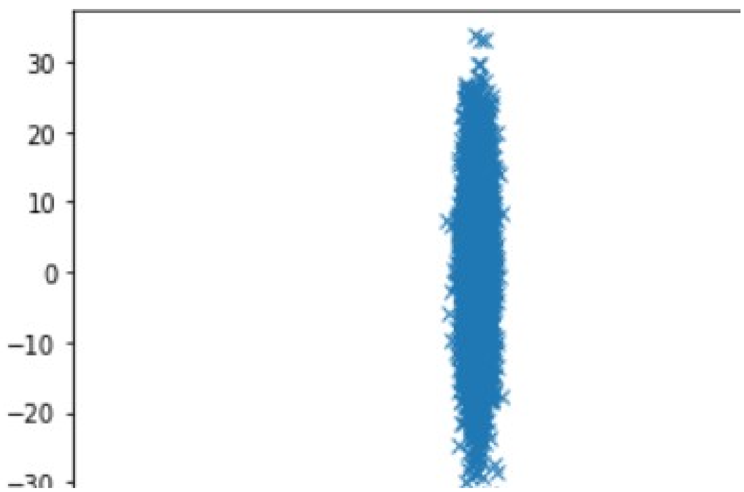
7. Explore how the empirical error does or does not approach the Bhattacharyya bound as follows:

(a) Write a procedure to generate sample points in d dimensions with a normal distribution having mean μ and covariance matrix Σ.

Code:

```
import matplotlib.pyplot as plt
mean = [0, 0]
cov = [[1, 0], [0, 100]]
x, y = np.random.multivariate_normal(mean, cov, 5000).T
plt.plot(x, y, 'x')
plt.axis('equal')
plt.show()
```

Output:



(b) Consider p(x|ω1) ~ N((1,0),I) and p(x|ω2) ~ N ((−1 0) , I) with P(ω1) = P(ω2) = 1/2. By inspection, state the Bayes decision boundary.

Code:

```
'''
state the Bayes decision boundary
P(ω1) = P(ω2) = 1/2
'''

import numpy as np
import matplotlib.pyplot as plt

def Bayes():
    '''
    P(x|ω1) = N(x;μ1,Σ1)
    P(x|ω2) = N(x;μ2,Σ2)
    '''
    # P(x|ω1) = N(x;μ1,Σ1)
    x1 = np.linspace(-10,10,1000)
```

```
m1 = 0
s1 = [[1,0],[0,1]]
y1 = np.exp(-0.5*(x1-m1)@np.linalg.inv(s1)@(x1-m1))
# P(x|ω2) = N(x;μ2,Σ2)
x2 = np.linspace(-10,10,1000)
m2 = 1
s2 = [[1,0],[0,1]]
y2 = np.exp(-0.5*(x2-m2)@np.linalg.inv(s2)@(x2-m2))
# P(x|ω1) = P(x|ω2)
x = np.linspace(-10,10,1000)
y = y1/y2
plt.plot(x,y)
plt.xlabel('x')
plt.ylabel('P(x|ω1)/P(x|ω2)')
plt.title('Bayes decision boundary')
plt.show()
```

(c) Generate n = 100 points (50 for ω1 and 50 for ω2) and calculate the empirical error.

Code:

```
import numpy as np
n=100
points = np.random.uniform(-1, 1, (n, 2))
# Calculate the empirical error
empirical_error = np.sum(np.sign(points[:, 0]**2 + points[:, 1]**2 - 0.6) != np.sign(points[:, 1])) / n
print(empirical_error)
```

Ouput:

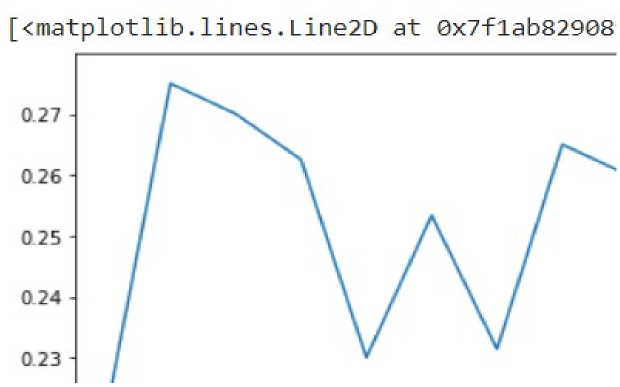
```
n=100
points = np.random.uniform(-1, 1, (n, 2))
    # Calculate the empirical error
empirical_error = np.sum(np.sign(points[:, 0]**2 + points[:, 1]**2 - 0.6) != np.sig
print(empirical_error)
```

(d) Repeat for increasing values of n, 100 ≤ n ≤ 1000, in steps of 100 and plot your empirical error.

Code:

```
import numpy as np
n=100
points = np.random.uniform(-1, 1, (n, 2))
# Calculate the empirical error
empirical_error = np.sum(np.sign(points[:, 0]**2 + points[:, 1]**2 - 0.6) != np.sign(points[:, 1])) / n
print(empirical_error)
```

Output:



(e) Discuss your results. In particular, is it ever possible that the empirical error is greater than the Bhattacharyya or Chernoff bound?

Inference:
Chernoff bound is never looser than the Bhattacharya bound. Here Chernoff bound is at $\beta^* = 0.66$ and is slightly tighter than the Bhattacharya bound ($\beta = 0.5$)