Statistical and Syntactic Pattern Recognition (Comp5107)

Tansin Jahan

Report on Assignment 3

For this assignment, M1 and M2 given as input is following -

M1 = [4,5,7]

M2 = [-9,-12,-11]

And the covariances are following :

$\Sigma 1$ = [[a^2 βab αac], [βab b^2 βbc], [αac βbc c^2]]

$\Sigma 2$ = [[c^2 αbc βac], [αbc b^2 αab], [βac αab a^2]]

Where values of the above variables are - a = 2, b=3, c=4 and α = 0.1 , β = 0.2

Answer to Q(a) :

```
def mean_input():
   print("Please give mean for first class in this format x,y,z")
   m1 = input()
   m1 = np.asarray(m1)
   print("Please give mean for second class in this format x,y,z")
   m2 = input()
   m2 = np.asarray(m2)
   return m1,m2
```

[----------generate points----------]

```
print("This is 200 points of X1:")
X1 = gn.generate_points(1)

print("This is 200 points of X2:")
X2 = gn.generate_points(2)
```

[----------plot------------]

```
plt.scatter(X1_0_1[:, [0]], X1_0_1[:, [1]], c='red')
plt.scatter(X2_0_1[:, [0]], X2_0_1[:, [1]], c='blue')

plt.title("X1 - X2 domain")
plt.show()

plt.scatter(X1_0_2[:, 0], X1_0_2[:, 1], c='red')
plt.scatter(X2_0_2[:, 0], X2_0_2[:, 1], c='blue')

plt.title("X1 - X3 domain")
plt.show()
```

Answer to Q(b) :

Given Bayes Optimal function form -

$$X^T A X + B^T X + C > 0$$

For the known M1, M2 , Σ1 and Σ2 , the following formula is used to create A, B and C -

$A = (\Sigma_2^{-1} - \Sigma_1^{-1})/2$ $\quad$ $B^T = (M_1^T \Sigma_1^{-1} - M_2^T \Sigma_2^{-1})$ $\quad$ $C = \log(P_1 / P_2) + \log(|\Sigma_2| / |\Sigma_1|)$

And using A, B, C matrix the roots to generate optimal Bayes Function is calculated. Then setting x_3 = 0 and solving x_1 and x_2 the quadratic equation for (X_1- X_2) domain is generated -

$$a_{22}x_2^2 + (a_{12}x_1 + a_{21}x_1 + b_{12})\, x_2 + (a_{11}x_1^2 + b_{11}x_1 + C) = 0$$

In the same way the quadratic equation for (X_1 - X_3) domain is generated -

$$a_{33}x_3^2 + (a_{13}x_1 + a_{31}x_1 + b_{13})\, x_3 + (a_{11}x_1^2 + b_{11}x_1 + C) = 0$$

```
def discriminant_function_X1X2(A,B,C):
    root1 = np.array([])
    root2 = np.array([])
    points_x1 = np.array([])

    for x1 in np.arange(-15, 20, 0.1):
        # for X1 - X2 domain
        m = A[1][1]
        n = ((A[0][1] * x1) + (A[1][0] * x1)
+ B[1])
        o = A[0][0] * x1 * x1 + B[0] * x1 + C

        coef_array = np.array([m, n, o])
        r1, r2 = np.roots(coef_array)

        root1 = np.append(root1, r1)
        root2 = np.append(root2, r2)
        points_x1 = np.append(points_x1,
[x1])

    return root1, root2, points_x1
```

```
def discriminant_function_X1X3(A,B,C):
    root1 = np.array([])
    root2 = np.array([])
    points_x1 = np.array([])

    for x1 in np.arange(-15,10,0.1):
        #for X1 - X3 domain
        p = A[2][2]
        q = ((A[0][2] * x1)+ (A[2][0] *
x1) +B[2])
        r = A[0][0] * x1 *x1 + B[0] * x1 +
C


        coef_array = np.array([p,q,r])
        r1, r2 = np.roots(coef_array)

        root1 = np.append(root1,r1)
        root2 = np.append(root2,r2)
        points_x1 =
np.append(points_x1,[x1])

    return root1,root2,points_x1
```

[----------plot------------]

```
# X1-X2 domain

plt.scatter(Points_X1_X2[:], Root4[:], c='green')
plt.scatter(Points_X1_X2[:], Root3[:], c='green')
```

```
# X1-X3 domain

plt.scatter(Points_X1_X3[:], Root1[:], c='green')
plt.scatter(Points_X1_X3[:], Root2[:], c='green')
```

Answer to Q(c) :

Test points are generated by the following two methods -

def generate_testPoints_X1():
    test_Point_x1 = np.array([])
    for i in range(0,200):
        test_x1 = gn.generation_Of_X1()
        testx1_trans = np.transpose(test_x1)
        TP, TN = generate_classifier_x1 (test_x1,testx1_trans)
        test_Point_x1 = np.append(test_Point_x1,test_x1)
 	
    test_Point_x1 = test_Point_x1.reshape(200,3)
    v1_m,v1_c, diagonalize_x1 = gn.generation_of_V1()
    return TP,TN, test_Point_x1, v1_m,v1_c, diagonalize_x1
	def generate_testPoints_X2():
	    test_Point_x2 = np.array([])
	    for i in range(0, 200):
	        test_x2 = gn.generation_Of_X2()
	        testx2_trans= np.transpose(test_x2)
	        TP, TN = generate_classifier_x2 (test_x2,testx2_trans)
	        test_Point_x2 = np.append(test_Point_x2, test_x2)
	
	test_Point_x2 = test_Point_x2.reshape(200, 3)
	    v2_m, v2_c, diagonalize_x2 = gn.generation_of_V2()
	return TP, TN, test_Point_x2, v2_m, v2_c, diagonalize_x2

For each point, to classify them using the discriminant function, the discriminant function is calculated. If the value > 0 then the point is from class 1 otherwise if value < 0 then point is from class 2.

```
[-----------classification for testing points(X1) ---------]

def generate_classifier_x1(point,transpose_point):
    global true_positive_X1
    global true_negative_X1
    value = ((np.dot(np.dot(point, A), transpose_point)) + (np.dot(B,transpose_point)) + C)
    if value >0:
        true_positive_X1 = true_positive_X1 +1
    else:
        true_negative_X1 = true_negative_X1 + 1

    return true_positive_X1, true_negative_X1
```

Same way, discriminant function is calculated for X2 and classified the generated test points X2 . The confusion matrix for accuracy is -

	X1(truePositive)	X2(trueNegative)
X1(truePositive)	198	2
X2(trueNegative)	0	200

Accuracy : 99.5%

Answer to Q(d) :

[----------generation of diagonalized training points-----]

```
mean_V1,covariance_V1,V1 = gn.generation_of_V1()
mean_V2,covariance_V2,V2 = gn.generation_of_V2()
```

[----------- to plot ----------]

```
plt.scatter(V1_0_1[:, [0]], V1_0_1[:, [1]], c='red')
plt.scatter(V2_0_1[:, [0]], V2_0_1[:, [1]], c='blue')

plt.title("V1 - V2 domain")

plt.scatter(V1_0_2[:, 0], V1_0_2[:, 1], c='red')
plt.scatter(V2_0_2[:, 0], V2_0_2[:, 1], c='blue')

plt.title("V1 - V3 domain")
```

Answer to Q(e) :

While generating testing points in question number (c) for each class (X1 and X2), I have diagonalized those points into V1 and V2 domain as following -

```
v1_m,v1_c, diagonalize_x1 = gn.generation_of_V1()
v2_m,v2_c, diagonalize_x2 = gn.generation_of_V2()
```

Then in the transferred domain computed the optimal Bayes Discriminant function as follows -

[---transferred domain Bayes function with different (A,B,C)-----]

```
Root5,Root6,Points_V1_V2 = discriminant_function_X1X2(v_A,v_B,v_C)
Root7,Root8,Points_V1_V3 = discriminant_function_X1X3(v_A,v_B,v_C)
```

Answer to Q(f) :

[-----------classification in transferred domain (V1) ---------]

```
def generate_classifier_v1(point):
    global true_positive_V1
    global true_negative_V1
    r,c = point.shape
    for i in range(0,r):
            p = point[i]
            p_trans =np.transpose(p)

value = ((np.dot(np.dot(point[i], v_A), p_trans)) + (np.dot(v_B,p_trans)) + v_C)
            if value >0:
                true_positive_V1 = true_positive_V1 +1
            else:
                true_negative_V1 = true_negative_V1 + 1

    return true_positive_V1, true_negative_V1
```

Same way classification in transferred domain for V2 points are done using **generate_classifier_v2(point)** method and the confusion matrix is following -

	V1(truePositive)	**V2(trueNegative)**
V1(truePositive)	198	2
V2(trueNegative)	0	200

Accuracy : 99.75%

[-------------- Plots -------------]

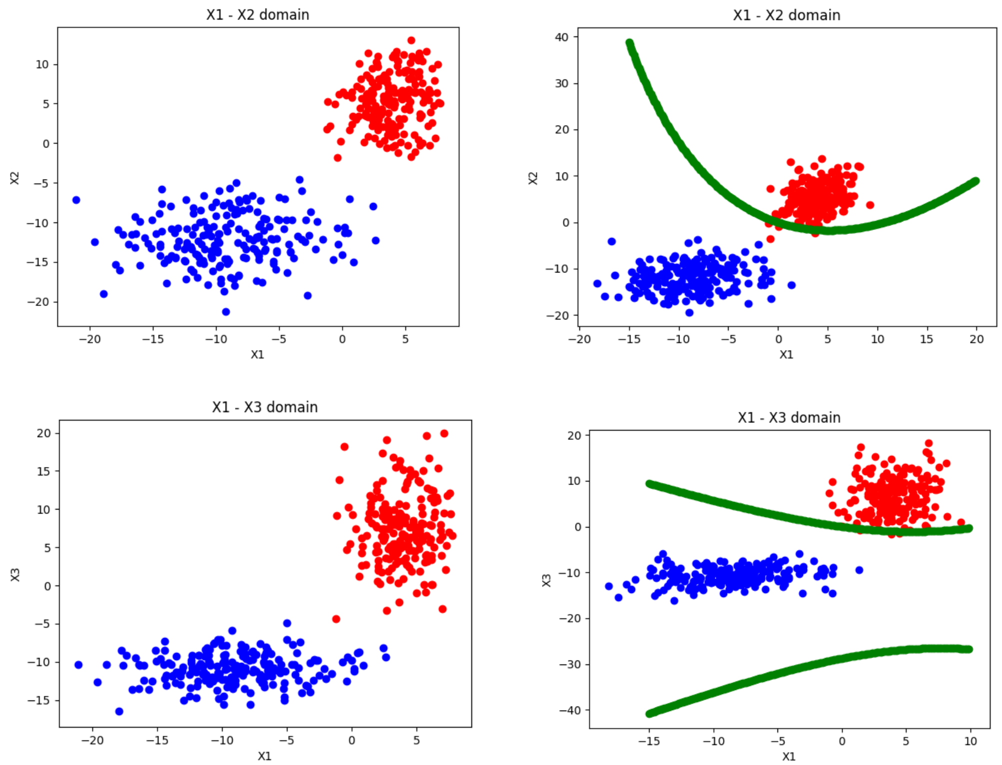

Fig : Before diagonalization of points in two classes

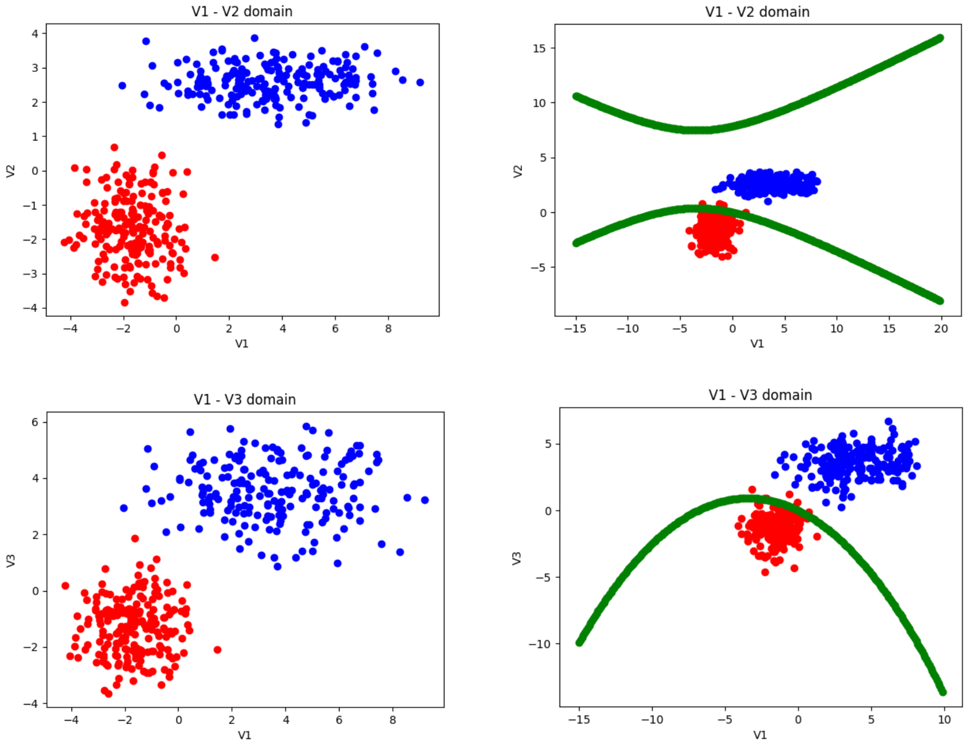

Fig : transfer domain (After diagonalization)