PATTERN RECOGNITION

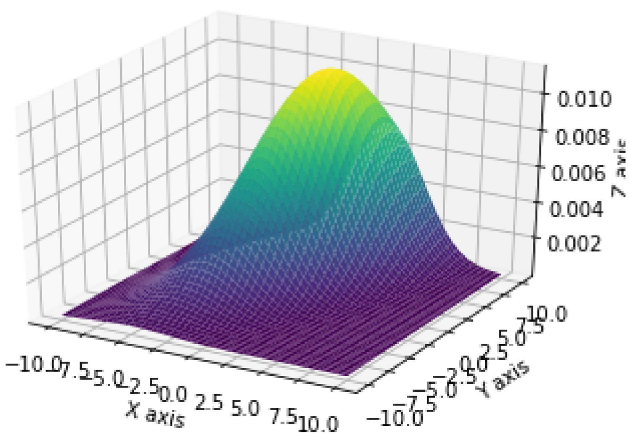
Assignment 1: Discriminant Functions

Submitted by: Vikas Garg, 17MCMC10, MCA

In [1]:
```
# Libraries needed for all questions

import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal
import numpy as np
import seaborn as sns
from mpl_toolkits.mplot3d import Axes3D

import math
from sklearn.metrics import confusion_matrix
from sklearn.datasets import load_iris
```

In [2]:
```
# procedure to generate random samples according to a normal distribution N(m
u,var) in d dimensions

def random_samples(m, v, s):

        if len(m) != 1:
                n = len(v)
                cov = [[0] * n] * n
                for i in range(0,n):
                        cov[i][i] = v[i]
                return multivariate_normal(m, cov, s)
        else:
                return np.random.normal(m, v, s)
```

In [3]:
```
# Example of generation random samples according to multivariate normal distri
bution

m = [0, 5]
v = [10, 30]
rv = random_samples(m, v, 50)

if(len(m) == 1):
        sns.kdeplot(rv)
        plt.show()
else:
        x = np.linspace(-10,10,500)
        y = np.linspace(-10,10,500)
        X, Y = np.meshgrid(x,y)
        pos = np.empty(X.shape + (2,))
        pos[:, :, 0] = X; pos[:, :, 1] = Y
        fig = plt.figure()
        ax = fig.gca(projection='3d')
        ax.plot_surface(X, Y, rv.pdf(pos),cmap='viridis',linewidth=0)
        ax.set_xlabel('X axis')
        ax.set_ylabel('Y axis')
        ax.set_zlabel('Z axis')
        plt.show()
```



In [4]:
```
# Read Synthetic data and Iris dataset

# Syntetic Data
filename = 'data_dhs_chap2.csv'
x = np.loadtxt(filename, delimiter=',', skiprows=1, usecols=[0,1,2])
y = np.loadtxt(filename, delimiter=',', skiprows=1, usecols=[3])

# Iris dataset
iris_data = load_iris()
```

In [6]:
```
# Mean and Covariance for the Synthetic Data

m1 = np.mean(x[y==0], axis=0)
m2 = np.mean(x[y==1], axis=0)
m3 = np.mean(x[y==2], axis=0)

cov1 = np.cov(x[y==0].T)
cov2 = np.cov(x[y==1].T)
cov3 = np.cov(x[y==2].T)

means = [m1, m2, m3]
cov = [cov1, cov2, cov3]
for i in range(3):
    print("\nCLASS [w{}]:\nMean:{}\nCovarience Matrix:\n{}".format(i+1, means[
i], cov[i]))
```

```
CLASS [w1]:
Mean:[-0.44  -1.749 -0.766]
Covarience Matrix:
[[14.38051111  7.69537778  4.12232222]
 [ 7.69537778 14.62312111  3.90684   ]
 [ 4.12232222  3.90684    19.72453778]]

CLASS [w2]:
Mean:[-0.543 -0.762 -0.542]
Covarience Matrix:
[[ 36.82933444   9.98092667 -16.36675111]
 [  9.98092667  13.16855111   0.40905111]
 [-16.36675111   0.40905111  18.42121778]]

CLASS [w3]:
Mean:[3.883 1.376 1.58 ]
Covarience Matrix:
[[ 8.30475667  7.44494667 13.14957778]
 [ 7.44494667  8.56044889 11.60861111]
 [13.14957778 11.60861111 47.28728889]]
```

In [60]:
```
# General procedures that will be usefull in exercise

# procedure for calculate univariate discriminant function
def uni_dis(x, m, sigma, pw):
        return -(0.5/sigma)*(x - m)*(x - m) - (0.5)*math.log(2*np.pi) - math.l
og(math.sqrt(sigma)) + math.log(pw)

# procedure for calculate multivariate discriminant function
def disc_fun(x, m, cov, pw):
        dim = x.shape[0]
        pw = np.array(pw)
        return -0.5*(np.dot(np.dot((x-m).T, np.linalg.inv(cov)), (x-m))) - (di
m/2)*math.log(2*math.pi) - 0.5*math.log(np.linalg.det(cov)) + math.log(pw)

# procedure for calculate error percentage in training data
def error(pred, y):
        return ((pred!=y).astype(int).sum()/pred.shape[0])*100

# Procedure for calculate Euledian Distance
def euledian(x1, x2):
        return math.sqrt(np.sum((x1 - x2)**2, axis=0))

# Procedure for calculate Mahalanobis distance
def mahalanobis(x, mu, cov):
        x = np.array(x)
        mu = np.array(mu)
        return math.sqrt(np.dot(np.dot((x-mu).T, np.linalg.inv(cov)), (x-mu)))
```

In [55]:
```
# Examples of Eucledian and Mahalanobis Distance

print('\nEucledian Distance between [1, 2, 1] and {}:'.format(m1), euledian(np
.array([1,2,1]), m1))

print('\nMahalanobis Distance between [1, 2, 1] and {}:'.format(m1), mahalanob
is(np.array([1,2,1]), m1, cov1))
```

```
Eucludian Distance between [1, 2, 1] and [-0.44  -1.749 -0.766]: 4.3871809855
53251

Mahalanobis Distance between [1, 2, 1] and [-0.44  -1.749 -0.766]: 1.01497062
11958083
```

In [21]:
```python
# procedure of compute the Decotomozer of 1 D feature space with 2 classes

def univariate(x, y, pw):

        x1 = x[:20,0]

        m = [np.mean(x1[:10]), np.mean(x1[10:20])]
        sigma = [np.cov(x1[0:10]), np.cov(x1[10:20])]

        g1x = uni_dis(x1, m[0], sigma[0], pw[0])
        g2x = uni_dis(x1, m[1], sigma[1], pw[1])

        pred = (g1x < g2x)

        return pred

''' As the rule of Discriminant Function Classification, points belong to that
class whose discriminant function in Maximum'''

pred = univariate(x, y, [0.5, 0.5])
cm = confusion_matrix(y[:20], pred)
err = error(pred, y[:20])
print('\n1.) Error of dicotomozer of 1 D feature Space: {}%'.format(err))
print('Confusion Matrix: \n', cm)
```

```
1.) Error of dicotomozer of 1 D feature Space: 30.0%
Confusion Matrix:
 [[7 3]
 [3 7]]
```

In [38]:
```
# procedure of compute the Decotomozer of 2 D and 3 D feature space with 2 cla
sses

def multivariate(x, y, pw):
        x = np.array(x)

        dim = x.shape[1]
        samples = x.shape[0]
        classes = len(pw)

        g1x = np.array([])
        g2x = np.array([])
        g3x = np.array([])

        m1 = np.array([np.mean(x[:10,0]), np.mean(x[0:10, 1])])
        m2 = np.array([np.mean(x[10:20,0]), np.mean(x[10:20, 1])])

        if dim == 3:
                m1  = np.concatenate((m1, np.array([ np.mean(x[0:10, 2]) ]) ),
axis=None)
                m2  = np.concatenate((m2, np.array([ np.mean(x[10:20, 2]) ])
), axis=None)

        sigma1 = np.cov(x[0:10, 0:dim].T)
        sigma2 = np.cov(x[10:20, 0:dim].T)

        if classes == 3:
                sigma3 = np.cov(x[20:30, 0:dim].T)
                m3 = np.array([np.mean(x[20:30,0]), np.mean(x[20:30, 1]), np.m
ean(x[20:30, 2]) ])

        for i in range(0,samples):
                g1 = disc_fun(np.array(x[i, 0:dim]).T, m1.T, sigma1, pw[0])
                g1x = np.append(g1x, g1)

                g2 = disc_fun(np.array(x[i, 0:dim]).T, m2.T, sigma2, pw[1])
                g2x = np.append(g2x, g2)

                if classes == 3:
                        g3 = disc_fun(np.array(x[i, 0:dim]).T, m3.T, sigma3, p
w[2])
                        g3x = np.append(g3x, g3)

        g = np.concatenate((g1x.reshape(samples, 1), g2x.reshape(samples, 1)),
axis = 1)
        if classes == 3:
                g = np.concatenate((g, g3x.reshape(samples, 1)), axis = 1)

        pred = np.argmax(g, axis=1)
        return g, pred

g, pred = multivariate(x[0:20, 0:2], y, [0.5, 0.5])
print('By the rule of maximum discriminant function, the classification result
s are \n', )
cm = confusion_matrix(y[:20], pred)
err = error(pred, y[:20])
```

```
print('\n2.) Error of dicotomozer of 2 D feature Space: {}%'.format(err))
print('Confusion Matrix: \n', cm)
print('\n   g1x          g2x        predictions')
print(np.concatenate((g, pred.reshape(20,1)), axis=1))

g, pred = multivariate(x[0:20, 0:3], y, [0.5, 0.5])
cm = confusion_matrix(y[:20], pred)
err = error(pred, y[:20])
print('\n3.) Error of dicotomozer of 3 D feature Space: {}%'.format(err))
print('Confusion Matrix: \n', cm)
print('\n   g1x          g2x        predictions')
print(np.concatenate((g, pred.reshape(20,1)), axis=1))
```

By the rule of maximum discriminant function, the classification results are

```
2.) Error of dicotomozer of 2 D feature Space: 45.0%
Confusion Matrix:
 [[5 5]
 [4 6]]

   g1x          g2x        predictions
[[-6.49946702 -7.58483495  0.        ]
 [-5.94764239 -5.92516495  1.        ]
 [-6.12052764 -6.83490421  0.        ]
 [-5.45251316 -6.08665872  0.        ]
 [-5.82019285 -5.75469687  1.        ]
 [-6.26833699 -6.23091865  1.        ]
 [-6.35354211 -6.10819916  1.        ]
 [-5.17023291 -5.58685586  0.        ]
 [-6.38464133 -6.19887214  1.        ]
 [-5.38183335 -5.9701421   0.        ]
 [-5.20531803 -5.53214525  0.        ]
 [-5.218594   -5.70862472  0.        ]
 [-6.95761363 -6.37241112  1.        ]
 [-7.08141622 -6.16002852  1.        ]
 [-7.28709203 -7.15698536  1.        ]
 [-5.6415364  -5.77979206  0.        ]
 [-7.92135535 -7.41278384  1.        ]
 [-7.09466434 -6.31866672  1.        ]
 [-8.81662903 -6.83009421  1.        ]
 [-6.80285064 -6.8097312   0.        ]]

3.) Error of dicotomozer of 3 D feature Space: 15.0%
Confusion Matrix:
 [[8 2]
 [1 9]]

   g1x          g2x        predictions
[[ -8.89663381  -9.88817976   0.        ]
 [ -8.38036882  -9.08684752   0.        ]
 [ -8.69639914 -10.4421242    0.        ]
 [ -8.11761734  -8.20447106   0.        ]
 [-10.03969958  -9.83941732   1.        ]
 [ -8.66474862  -9.08433345   0.        ]
 [ -8.83800477  -9.25207517   0.        ]
 [ -8.44677649 -10.14781384   0.        ]
 [ -9.08010492  -8.21150281   1.        ]
 [ -8.45356356 -10.70022425   0.        ]
 [ -7.58527732  -7.54439959   1.        ]
 [ -7.84666596  -7.8212055    1.        ]
 [ -9.41103361  -8.81419039   1.        ]
 [-10.22990214  -8.23655531   1.        ]
 [-10.85428301  -9.85730405   1.        ]
 [ -8.41469417 -10.03956186   0.        ]
 [-10.64358362  -9.66155324   1.        ]
 [-11.1907397   -8.75419976   1.        ]
 [-12.93266582  -9.07730019   1.        ]
 [ -9.3605496   -8.89743559   1.        ]]
```

In [46]:
```
# Prediction of all the three categories data with different prior probabiliti
es

g, pred = multivariate(x, y, [0.333, 0.333, 0.333])
cm = confusion_matrix(y, pred)
err = error(pred, y)
print('\nError of 3 class classification with prior probabilities {}: {}%'.for
mat([0.333, 0.333, 0.333], err))
print('Confusion Matrix: \n', cm)
print('\n   g1x          g2x          g3x            predictions')
print(np.concatenate((g, pred.reshape(30,1)), axis=1))

g, pred = multivariate(x, y, [0.8, 0.1, 0.1])
cm = confusion_matrix(y, pred)
err = error(pred, y)
print('\nError of 3 class classification with prior probabilities{}: {}%'.form
at([0.8, 0.1, 0.1], err))
print('Confusion Matrix: \n', cm)
print('\n   g1x          g2x          g3x            predictions')
print(np.concatenate((g, pred.reshape(30,1)), axis=1))
```

```
Error of 3 class classification with prior probabilities [0.333, 0.333, 0.33
3]: 20.0%
Confusion Matrix:
 [[7 1 2]
 [0 8 2]
 [1 0 9]]

   g1x          g2x          g3x           predictions
[[ -9.30309942 -10.29464536 -13.66957636   0.         ]
 [ -8.78683443  -9.49331313 -17.20010323   0.         ]
 [ -9.10286474 -10.84858981 -12.75336717   0.         ]
 [ -8.52408295  -8.61093667  -9.03023617   0.         ]
 [-10.44616519 -10.24588293 -21.68281389   1.         ]
 [ -9.07121423  -9.49079906  -7.20672275   2.         ]
 [ -9.24447038  -9.65854078 -21.1563346    0.         ]
 [ -8.85324209 -10.55427944 -10.22223407   0.         ]
 [ -9.48657053  -8.61796842  -7.83718581   2.         ]
 [ -8.86002917 -11.10668986  -9.52627117   0.         ]
 [ -7.99174293  -7.95086519 -10.97541334   1.         ]
 [ -8.25313156  -8.22767111  -7.62821476   2.         ]
 [ -9.81749922  -9.220656   -25.44942877   1.         ]
 [-10.63636775  -8.64302092 -33.11396526   1.         ]
 [-11.26074862 -10.26376966 -10.413165     1.         ]
 [ -8.82115978 -10.44602747  -7.07820914   2.         ]
 [-11.05004923 -10.06801885 -22.57699446   1.         ]
 [-11.59720531  -9.16066537 -13.27569705   1.         ]
 [-13.33913143  -9.48376579 -50.94996891   1.         ]
 [ -9.76701521  -9.3039012  -21.40615914   1.         ]
 [-10.39836091 -14.84120071  -7.3798732    2.         ]
 [ -9.47934024  -8.67869863  -7.81516221   2.         ]
 [-10.20455757 -12.14096513  -9.79122831   2.         ]
 [ -9.43364016 -11.13874045  -7.64375064   2.         ]
 [-11.46368732 -17.76796625  -8.53614798   2.         ]
 [-10.22563941  -9.30875735  -8.65976344   2.         ]
 [-10.18070546 -13.04267905  -7.39574187   2.         ]
 [ -8.1443291   -8.5551367   -8.86793269   0.         ]
 [ -9.84062087 -11.11310491  -8.17383895   2.         ]
 [ -9.43350722 -12.82521719  -7.93423058   2.         ]]

Error of 3 class classification with prior probabilities[0.8, 0.1, 0.1]: 50.
0%
Confusion Matrix:
 [[10  0  0]
 [ 8  2  0]
 [ 7  0  3]]

   g1x          g2x          g3x           predictions
[[ -8.42663018 -11.49761767 -14.87254867   0.         ]
 [ -7.91036519 -10.69628544 -18.40307554   0.         ]
 [ -8.22639551 -12.05156212 -13.95633947   0.         ]
 [ -7.64761371  -9.81390898 -10.23320847   0.         ]
 [ -9.56969595 -11.44885524 -22.88578619   0.         ]
 [ -8.19474499 -10.69377136  -8.40969505   0.         ]
 [ -8.36800114 -10.86151308 -22.3593069    0.         ]
 [ -7.97677286 -11.75725175 -11.42520637   0.         ]
 [ -8.61010129  -9.82094072  -9.04015811   0.         ]
 [ -7.98355993 -12.30966217 -10.72924348   0.         ]
```

```
 [ -7.11527369  -9.1538375  -12.17838565   0.        ]
 [ -7.37666233  -9.43064341  -8.83118707   0.        ]
 [ -8.94102998 -10.4236283  -26.65240108   0.        ]
 [ -9.75989851  -9.84599322 -34.31693756   0.        ]
 [-10.38427938 -11.46674196 -11.61613731   0.        ]
 [ -7.94469054 -11.64899977  -8.28118144   0.        ]
 [-10.17357999 -11.27099116 -23.77996677   0.        ]
 [-10.72073607 -10.36363768 -14.47866936   1.        ]
 [-12.4626622  -10.6867381  -52.15294121   1.        ]
 [ -8.89054597 -10.50687351 -22.60913144   0.        ]
 [ -9.52189167 -16.04417302  -8.5828455    2.        ]
 [ -8.602871    -9.88167093  -9.01813451   0.        ]
 [ -9.32808833 -13.34393744 -10.99420062   0.        ]
 [ -8.55717092 -12.34171276  -8.84672295   0.        ]
 [-10.58721808 -18.97093856  -9.73912028   2.        ]
 [ -9.34917017 -10.51172966  -9.86273574   0.        ]
 [ -9.30423623 -14.24565135  -8.59871418   2.        ]
 [ -7.26785986  -9.75810901 -10.070905     0.        ]
 [ -8.96415163 -12.31607722  -9.37681125   0.        ]
 [ -8.55703798 -14.0281895   -9.13720288   0.        ]]
```

In [50]:

```
# prediction on Test data with priors [0.333, 0.333, 0.333] and [0.8, 0.8, 0.
8] using each of the category means

test_data = [[1, 2 ,1],
                         [5, 3, 2],
                         [0, 0, 0],
                         [1, 0, 0]]
test_samples = len(test_data)
test_data = np.array(test_data)

priors = [[0.333, 0.333, 0.333],
                        [0.8, 0.1, 0.1]]

for i in range(len(priors)):
        g1x = np.array([])
        g2x = np.array([])
        g3x = np.array([])
        print('\nTest data Classification with prior probabilities {} are: '.f
ormat(priors[i]))
        for j in range(test_samples):
                g1 = disc_fun(test_data[j], m1, cov1, priors[i][0])
                g1x = np.append(g1x, g1)

                g2 = disc_fun(test_data[j], m2, cov2, priors[i][1])
                g2x = np.append(g2x, g2)

                g3 = disc_fun(test_data[j], m3, cov3, priors[i][2])
                g3x = np.append(g3x, g3)

        g_test = np.concatenate((g1x.reshape(test_samples, 1), g2x.reshape(tes
t_samples, 1), g3x.reshape(test_samples, 1)), axis = 1)
        pred = np.argmin(g_test, axis=1)

        for k in range(test_samples):
                print('{} belongs to class {}'.format(test_data[k], pred[k]))
```

```
Test data Classification with prior probabilities [0.333, 0.333, 0.333] are:
[1 2 1] belongs to class 2
[5 3 2] belongs to class 1
[0 0 0] belongs to class 2
[1 0 0] belongs to class 1

Test data Classification with prior probabilities [0.8, 0.1, 0.1] are:
[1 2 1] belongs to class 2
[5 3 2] belongs to class 1
[0 0 0] belongs to class 2
[1 0 0] belongs to class 1
```

In [52]:

```
# prediction on Test data with priors [0.333, 0.333, 0.333] and [0.8, 0.8, 0.
8] using Mahalanobis Distance

d1x = np.array([])
d2x = np.array([])
d3x = np.array([])
for i in range(test_samples):
        d1 = mahalanobis(test_data[i], m1, cov1)
        d1x = np.append(d1x, d1)

        d2 = mahalanobis(test_data[i], m2, cov2)
        d2x = np.append(d2x, d2)

        d3 = mahalanobis(test_data[i], m2, cov3)
        d3x = np.append(d3x, d3)

d = np.concatenate((d1x.reshape(test_samples, 1), d2x.reshape(test_samples, 1
), d3x.reshape(test_samples, 1)), axis=1)
pred = np.argmin(d, axis=1)
for k in range(test_samples):
                print('{} belongs to class {}'.format(test_data[k], pred[k]))
```

```
[1 2 1] belongs to class 1
[5 3 2] belongs to class 0
[0 0 0] belongs to class 1
[1 0 0] belongs to class 1
```

In [54]:
```
# procedure for Discriminant Function for all three cases on IRIS Dataset

def pred_iris(x, y, pw, case):
        x = np.array(x)

        dim = x.shape[1]
        samples = x.shape[0]
        classes = len(pw)

        g1x = np.array([])
        g2x = np.array([])
        g3x = np.array([])

        m1 = np.mean(x[y==0], axis=0)
        m2 = np.mean(x[y==1], axis=0)
        m3 = np.mean(x[y==2], axis=0)

        sigma1 = np.cov(x[y==0].T)
        sigma2 = np.cov(x[y==1].T)
        sigma3 = np.cov(x[y==2].T)

        if case == 1:
                sigma = ((np.diag(((sigma1+sigma2+sigma3)/3)*np.eye(dim)).sum
())/dim)*np.eye(dim)
                sigma1=sigma2=sigma3=sigma

        if case == 2:
                sigma1=sigma2=sigma3=((sigma1+sigma2+sigma3)/3)

        for i in range(samples):
                g1 = disc_fun(x[i], m1, sigma1, pw[0])
                g1x = np.append(g1x, g1)

                g2 = disc_fun(x[i], m2, sigma2, pw[1])
                g2x = np.append(g2x, g2)

                g3 = disc_fun(x[i], m3, sigma3, pw[2])
                g3x = np.append(g3x, g3)

        g = np.concatenate((g1x.reshape(samples, 1), g2x.reshape(samples, 1),
g3x.reshape(samples, 1)), axis = 1)
        pred = np.argmax(g, axis=1)
        return pred

for case in range(1,4):
        pred = pred_iris(iris_data['data'], iris_data['target'], [0.333, 0.333
, 0.333], case)
        cm = confusion_matrix(iris_data['target'], pred)
        print('\nConfusion Matrix for case',case, ' \n',cm)
```

```
Confusion Matrix for case 1
 [[50  0  0]
 [ 0 46  4]
 [ 0  7 43]]

Confusion Matrix for case 2
 [[50  0  0]
 [ 0 48  2]
 [ 0  1 49]]

Confusion Matrix for case 3
 [[50  0  0]
 [ 0 48  2]
 [ 0  1 49]]
```

In [ ]: