Generating Random Samples

We will now generate samples of the Gaussian Distribution with $\mathcal{N}(\mu,\Sigma).$ For this we first need to generate the μ and Σ matrices.

In [1]:

```
import numpy as np
import sklearn.datasets as skdataset
```

In [2]:

```
def getMeanAndCovMatrices(dim):
    cov_mat=skdataset.make_spd_matrix(dim)
    mean_vec=np.random.randn(1,dim)[0]
    return mean_vec, cov_mat
```

In [3]:

```
def generateRandomSamples(dim,num_of_samples):
    mean_vec, cov_mat = getMeanAndCovMatrices(dim)
    return np.random.multivariate_normal(mean_vec, cov_mat,(num_of_samples)).reshape(dim,num_of_samples)
```

Generating random samples of 5 dimension

In [4]:

```
generateRandomSamples(5,7) #generates 7 samples of whose dimension = 5
```

Out[4]:

```
array([[-0.71916817, -0.47791457, -0.35816703, -0.91663054, -0.55005985,
        -0.37910887, -0.83788672],
       [ 0.66560852, -0.14836529, -1.23325409, -1.17185418, -1.36653755,
         0.11842697, -0.31956973],
       [ 1.19289474, -0.03016004, -0.30509029, -0.26879544,  0.18675589,
        -2.26668049, -0.48347193],
       [ 1.12593443,  3.67953803,  2.58270316, -4.04189417,  0.57756252,
        -0.1774156 ,  0.92706617],
       [ 1.30659482, -0.03251858,  0.01434854,  0.78117878,  1.03329985,
         0.64618492, -1.89288452]])
```

Procedure to claculate Discriminant Function

For a given Normal Distribution $\mathcal{N}(\mu,\Sigma).$ and Prior probability $P(\omega_i)$ the discriminant function is given by the equation

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

In [5]:

```
def discriminant_function(dimension,x_vec,mean_vec,cov_mat,prior):
    if dimension > 1:
        cov_det = np.linalg.det(cov_mat)
        return ((-1 / 2) * mahalanobis(x_vec,mean_vec,cov_mat) \
                - (dimension / 2) * np.log(2 * np.pi) \
                - (1 / 2) * np.log(cov_det) + np.log(prior))
    else:
        return ((-1 / 2) * euclidean(x_vec,mean_vec) / cov_mat \
                - (1 / 2) * np.log(2 * np.pi * cov_mat) \
                + np.log(prior))
```

Mahalanobis Distance

In [6]:

```
def mahalanobis(x_vec,mean_vec,cov_mat):
    difference = x_vec - mean_vec
    cov_inv = np.linalg.inv(cov_mat)
    return np.dot(np.dot(difference.T,cov_inv),difference)
```

Euclidean Distance

In [7]:

```
def euclidean(p1,p2):
    return mahalanobis(p1,p2,np.identity(p1.shape[0]))
```

Reading dataset from csv file

In [8]:

```
data = np.genfromtxt("./synthetic_data.csv",delimiter=',', skip_header=1)
class1_data=data[:10]
class2_data=data[10:20]
class3_data=data[20:30]
```

Univarite and Multivariate Excersises: (2a , 2b, 2c, 2d)

Given prior probabilities are $P(\omega_1)=P(\omega_2)=0.5$

In [9]:

```
def dichotomiser(feature_dim, features, labels, prior1, prior2):
    class1_mean = np.mean(features[:10],axis=0).reshape(feature_dim,1)
    class2_mean = np.mean(features[10:20],axis=0).reshape(feature_dim,1)

    if(feature_dim > 1):
        class1_covariance = np.cov(features[:10].T)
        class2_covariance = np.cov(features[10:20].T)
    else:
        class1_covariance = np.var(features[:10],axis=0)
        class2_covariance = np.var(features[10:20],axis=0)

    class1_prior = prior1
    class2_prior = prior2

    prediction = []
    for feature in features:
        feature = feature.reshape(feature_dim,1)

        g1 = discriminant_function(feature_dim, feature, class1_mean, class1_covariance,class1_prior)
        g2 = discriminant_function(feature_dim, feature, class2_mean, class2_covariance,class2_prior)

        if(g1 - g2 > 0):
            prediction.append(0)
        elif(g1 - g2 < 0):
            prediction.append(1)
    print_predictions(labels, prediction)
    print("The Empirical Training error is : ", empirical_error(labels, prediction), "%")

def print_predictions(actual,prediction):
    print("Actual\t\t:\t", actual)
    print("Predicted\t:\t", prediction)


def empirical_error(actual,prediction):
    error_count=0
    for i,val in enumerate(prediction):
        if(actual[i] - val):
            error_count += 1
    return (error_count/actual.shape[0]) * 100
```

In [10]:

```
dichotomiser(1, data[:20,0], data[:20,3], 0.5, 0.5)
```

```
Actual          :        [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
Predicted       :        [1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1]
The Empirical Training error is :  35.0 %
```

In [11]:

```
dichotomiser(2, data[:20,:2], data[:20,3], 0.5, 0.5)
```

```
Actual          :        [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
Predicted       :        [0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0]
The Empirical Training error is :  45.0 %
```

In [12]:

```
dichotomiser(3, data[:20,:3], data[:20,3], 0.5, 0.5)
```

```
Actual          :        [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
Predicted       :        [0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1]
The Empirical Training error is :  15.0 %
```

2e. Claculating the test points Mahalanobis Distances

In [13]:

```
test_points = np.genfromtxt("./test-points.csv",delimiter=",",skip_header=1,dtype=int)
test_points
```

Out[13]:

```
array([[1, 2, 1],
       [5, 3, 2],
       [0, 0, 0],
       [1, 0, 0]])
```

In [14]:

```
feature_dim=3

class1_mean = np.mean(class1_data[:,:3],axis=0).reshape(feature_dim,1)
class2_mean = np.mean(class2_data[:,:3],axis=0).reshape(feature_dim,1)
class3_mean = np.mean(class3_data[:,:3],axis=0).reshape(feature_dim,1)

class1_covariance = np.cov(class1_data[:,:3].T)
class2_covariance = np.cov(class2_data[:,:3].T)
class3_covariance = np.cov(class3_data[:,:3].T)

prediction = []
for test_point in test_points:
    test_point=test_point.reshape(feature_dim,1)
    distances = []
    distances.append(mahalanobis(test_point,class1_mean,class1_covariance))
    distances.append(mahalanobis(test_point,class2_mean,class2_covariance))
    distances.append(mahalanobis(test_point,class3_mean,class3_covariance))
    prediction.append(distances.index(min(distances)))
print("Predicted classes based on Mahalanobis distance are : ",prediction)
```

```
Predicted classes based on Mahalanobis distance are :  [1, 2, 1, 1]
```

2f. With Change in Prior Probabilities

Now considering the prior probabilities of each of the class as $P(\omega_1) = 0.8, P(\omega_2)=0.1, P(\omega_3)=0.1$, predicting the class based on Mahalanobis distance has no effect. But considering the maximum value of LDF we get all the test points in class '0'.

In [15]:

```
prediction = []
for test_point in test_points:
    test_point=test_point.reshape(feature_dim,1)
    values = []
    values.append(discriminant_function(feature_dim,test_point,class1_mean,class1_covariance,0.8))
    values.append(discriminant_function(feature_dim,test_point,class2_mean,class2_covariance,0.1))
    values.append(discriminant_function(feature_dim,test_point,class3_mean,class3_covariance,0.1))
    prediction.append(values.index(max(values)))
print("Predicted classes are : ",prediction)
```

```
Predicted classes are :  [0, 0, 0, 0]
```

Iris Dataset

In [16]:

```
iris_dataset = np.genfromtxt("iris.csv",delimiter=",")
labels=["Iris-setosa","Iris-versicolor","Iris-virginica"]

iris_class1 = np.insert(iris_dataset[:50,:4],4,0,axis=1)
iris_class2 = np.insert(iris_dataset[50:100,:4],4,1,axis=1)
iris_class3 = np.insert(iris_dataset[100:150,:4],4,2,axis=1)
```

In [17]:

```
feature_dim=4
iris_samples1 = iris_class1[:,:4]
iris_samples2 = iris_class2[:,:4]
iris_samples3 = iris_class3[:,:4]

iris_class1_mean = np.mean(iris_samples1,axis=0).reshape(feature_dim,1)
iris_class2_mean = np.mean(iris_samples2,axis=0).reshape(feature_dim,1)
iris_class3_mean = np.mean(iris_samples3,axis=0).reshape(feature_dim,1)

#Case III Covariance matrices:
iris_class1_covariance = np.cov(iris_samples1.T)
iris_class2_covariance = np.cov(iris_samples2.T)
iris_class3_covariance = np.cov(iris_samples3.T)

#Case I Covariance matrix:
var1 = np.mean(iris_class1_covariance.diagonal())
var2 = np.mean(iris_class2_covariance.diagonal())
var3 = np.mean(iris_class3_covariance.diagonal())
average_var = (var1 + var2 + var3)/3
cov_mat1 = average_var*np.identity(feature_dim)

#Case II Covariance matrix:
cov_mat2 = (1/3 )* (iris_class1_covariance + iris_class2_covariance + iris_class3_covariance)
```

Iris Data Case 1

The LDF for Case 1 is given by the equation:

$g_i(x) = W_i^Tx+W_{i0} ------- (1)$

where $W_i= \frac{\mu_i}{\sigma^2} \implies W_i^T= \frac{\mu_i^T}{\sigma^2}$

and $W_{i0}= \frac{-\mu_i^T\mu_i}{2\sigma^2}+ ln(P(\omega_i))$

Calculating W_i^T

In [18]:

```
w1_t = iris_class1_mean.T / average_var
w2_t = iris_class2_mean.T / average_var
w3_t = iris_class3_mean.T / average_var
print(w1_t,w2_t,w3_t,sep="\n")
```

```
[[32.93023131 22.48412517  9.63041523  1.6050692 ]]
[[39.04791311 18.22148237 28.02292956  8.72263019]]
[[43.33686853 19.56342547 36.52190256 13.32733692]]
```

Calculating W_{i0}

In [19]:

```
w10 = (- 0.5) * (np.dot(iris_class1_mean.T,iris_class1_mean) / average_var) + np.log(1/3)
w20 = (- 0.5) * (np.dot(iris_class2_mean.T,iris_class2_mean) / average_var) + np.log(1/3)
w30 = (- 0.5) * (np.dot(iris_class3_mean.T,iris_class3_mean) / average_var) + np.log(1/3)
print(w10,w20,w30,sep="\n")
```

```
[[-129.19363357]]
[[-207.70151527]]
[[-287.82646472]]
```

By substituting the values of W_i^T and W_{i0} in equation (1) we get the Case 1 LDF for Iris dataset

$g_1(x) = 32.93* x_1 + 22.48 * x_2 + 9.63 * x_3 + 1.60 *x_4 - 129.19$

$g_2(x) = 39.05* x_1 + 18.22 * x_2 + 28.02 * x_3 + 8.72 *x_4 - 207.70$

$g_3(x) = 43.34* x_1 + 19.56 * x_2 + 36.52 * x_3 + 13.33 *x_4 - 287.83$

Iris Data Case 2

The LDF for Case 2 is given by the equation:

$g_i(x) = W_i^Tx+W_{i0} ------------ (2)$

where $W_i= \Sigma^{-1}\mu_i \implies W_i^T= \mu_i^T(\Sigma^{-1})^T$

and $W_{i0}= \frac{-1}{2}\mu_i^T\Sigma^{-1}\mu_i+ ln(P(\omega_i))$

Calculating W_i^T

In [20]:

```
w1_t = np.dot(iris_class1_mean.T,np.linalg.inv(cov_mat2).T)
w2_t = np.dot(iris_class2_mean.T,np.linalg.inv(cov_mat2).T)
w3_t = np.dot(iris_class3_mean.T,np.linalg.inv(cov_mat2).T)
print(w1_t,w2_t,w3_t,sep="\n")
```

```
[[ 23.46638411  23.56828007 -16.20296668 -18.02533894]]
[[15.70349917  6.95425598  5.28429325  6.29830031]]
[[12.49061995  3.44398145 12.82207622 21.06272174]]
```

Calculatin W_{i0}

In [21]:

```
w10 = (- 0.5) * np.dot(np.dot(iris_class1_mean.T,np.linalg.inv(cov_mat2)),iris_class1_mean) + np.log(1/3)
w20 = (- 0.5) * np.dot(np.dot(iris_class2_mean.T,np.linalg.inv(cov_mat2)),iris_class2_mean) + np.log(1/3)
w30 = (- 0.5) * np.dot(np.dot(iris_class3_mean.T,np.linalg.inv(cov_mat2)),iris_class3_mean) + np.log(1/3)
print(w10,w20,w30,sep="\n")
```

```
[[-86.05349939]]
[[-72.76956007]]
[[-104.2945355]]
```

By substituting the values of W_i^T and W_{i0} in equation (2) we get the Case 2 LDF for Iris dataset

$g_1(x) = 23.46* x_1 + 23.57 * x_2 - 16.20 * x_3 - 18.025 *x_4 - 86.05$

$g_2(x) = 15.70* x_1 + 6.95 * x_2 + 5.28 * x_3 + 6.29 *x_4 - 72.77$

$g_3(x) = 12.49* x_1 + 3.44 * x_2 + 12.82* x_3 + 21.06*x_4 - 104.294$

Iris Case 3

The QDF for Case 3 is given by the equation:

$ g_i(x) = x^tW_ix + w_i^tx + w_{i0} ------ (3) $

Where,

$ W_i = -\frac{1}{2}\Sigma_i $

$ w_i = \Sigma_i^{-1}\mu_i $

$ w_{i0} = -\frac{1}{2}\mu^t_i\Sigma_i^{-1}\mu_i -\frac{1}{2}\ln \lvert \Sigma_i \rvert + \ln(P(\omega_i)) $

Calculating ${W_i}$

In [22]:

```
W1 = (- 0.5) * iris_class1_covariance
W2 = (- 0.5) * iris_class2_covariance
W3 = (- 0.5) * iris_class3_covariance
print(W1,W2,W3,sep="\n\n")
```

```
[[-0.06212449 -0.05014898 -0.00806939 -0.00527347]
 [-0.05014898 -0.0725898  -0.00584082 -0.00571837]
 [-0.00806939 -0.00584082 -0.01505306 -0.00284898]
 [-0.00527347 -0.00571837 -0.00284898 -0.00574694]]

[[-0.13321633 -0.04259184 -0.09144898 -0.0278898 ]
 [-0.04259184 -0.04923469 -0.04132653 -0.02060204]
 [-0.09144898 -0.04132653 -0.11040816 -0.03655102]
 [-0.0278898  -0.02060204 -0.03655102 -0.01955306]]

[[-0.20217143 -0.04688163 -0.1516449  -0.02454694]
 [-0.04688163 -0.05200204 -0.0356898  -0.02381429]
 [-0.1516449  -0.0356898  -0.15229388 -0.02441224]
 [-0.02454694 -0.02381429 -0.02441224 -0.03771633]]
```

Calculating w_i

In [23]:

```
w1_t = np.dot(iris_class1_mean.T,np.linalg.inv(iris_class1_covariance).T)
w2_t = np.dot(iris_class2_mean.T,np.linalg.inv(iris_class2_covariance).T)
w3_t = np.dot(iris_class3_mean.T,np.linalg.inv(iris_class3_covariance).T)
print(w1_t,w2_t,w3_t,sep="\n")
```

```
[[ 44.6351704   -7.71338075  33.07861577 -28.45246274]]
[[ 18.0128645   15.96070005   3.26878502 -14.71255747]]
[[ 7.37247478 13.2452613   6.23406948  9.66197608]]
```

Calculating W_{i0}

In [24]:

```
w10 = (- 0.5 * np.dot(np.dot(iris_class1_mean.T,np.linalg.inv(iris_class1_covariance)),iris_class1_mean)) -
(0.5 * np.log(np.linalg.det(iris_class1_covariance))) + np.log(1/3)
w20 = (- 0.5 * np.dot(np.dot(iris_class2_mean.T,np.linalg.inv(iris_class2_covariance)),iris_class2_mean)) -
(0.5 * np.log(np.linalg.det(iris_class2_covariance))) + np.log(1/3)
w30 = (- 0.5 * np.dot(np.dot(iris_class3_mean.T,np.linalg.inv(iris_class3_covariance)),iris_class3_mean)) -
(0.5 * np.log(np.linalg.det(iris_class3_covariance))) + np.log(1/3)
print(w10,w20,w30,sep="\n")
```

```
[[-113.86317185]]
[[-68.43728769]]
[[-67.7090772]]
```

By substituting these values of W_i and w_i and W_{i0} in equation 3 we get the Quadratic Discriminant Function for Iris dataset