Pattern Recognition Assignment No:1
Code & Output
18MCMI09
Krishnan.S

All the algorithms were implemented using Python3. The libraries that were used were Numpy, Matplotlib and some standard python libraries like math and copy.

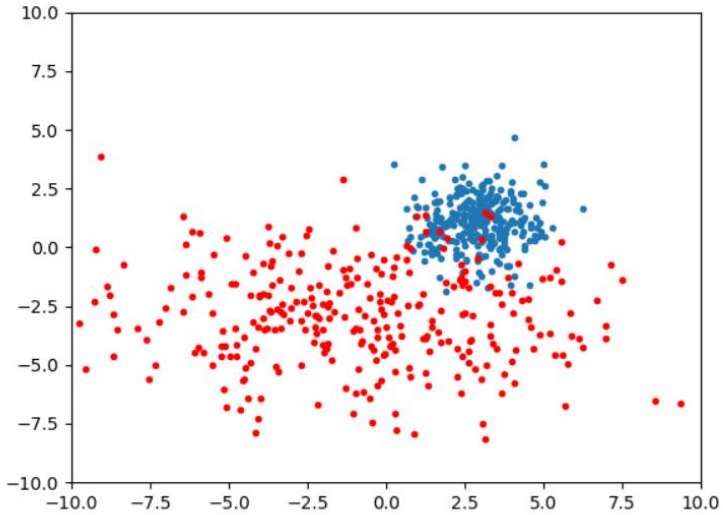
1.
A) Gaussian samples were generated using the Box Muller Transform. Two functions were written, one for univariate samples and one for multivariate samples. The dimension d has to be passed in the second function to indicate how many times a univariate sample must be picked to generate a d-dimensional sample.

Code:

```
def UVGaussianSample( mean, variance ):
    u1 = np.random.random()
    u2 = np.random.random()
    z0 = sqrt(-2*log(u1)) * cos(2*pi*u2) #Or sin
    return z0*variance + mean

def MVGaussianSample( mean, covariance, d ):
    Z = []
    for i in range(d):
        u1 = np.random.random()
        u2 = np.random.random()
        Z.append(sqrt(-2*log(u1)) * cos(2*pi*u2))  #Or sin
    Z = np.matmul(covariance, Z) + mean
    return Z
```

300 two dimensional random samples from two different distributions were plotted to show that they are Gaussian in nature. The first distribution has mean (3,1) and covariance ((1,0),(0,1)) and the second has mean (-1,-3) and covariance ((-4,0),(0,2)). Blue is the first, red is the second.

Output:



B) Two functions were written for the DF; one for univariate and one for multivariate. The functions use the distance functions in the next two subdivisions. Two numpy functions are used here. One to find the inverse of the covariance matrix (np.linalg.pinv). It calculates a pseudo inverse if the matrix is singular. The second function finds the determinant of the covariance matrix (np.linalg.det). linalg is the linear algebra module of numpy.
Log here calculates natural log.

Code:

```
def UVDF( x, mean, variance, prior ):
    A = -0.5*EuclideanU(x, mean)/variance
    B = -0.5*log(2*pi) -0.5*log(variance) + log(prior)
    return A+B

def MVDF( x, mean, covariance, prior ):
    d = len(x)
    covInv = np.linalg.pinv(covariance)
    covDet = np.linalg.det(covariance)

    A = -0.5*(Mahalanobis(x, mean, covariance)**2)
    B = -0.5*d*log(2*pi) -0.5*log(covDet) + log(prior)
    return A+B

x = np.array([0.3,0.5,0.4])
mean = np.array([0.1,0.4,0.8])
covariance = np.array([ [3, 0, 0], [0, 3, 0], [0, 0, 3]])
prior = 1/4

print("Input vector x :", x)
print("Mean : ", mean)
print("Covariance :")
print(covariance)
print("Prior : ", prior )
print("\nResult of DF on passing x : ", MVDF(x,mean,covariance,prior))
```

Output:

```
F:\CollegeStuff\Sem2\PR\Assg1>python Extra.py
Input vector x : [0.3 0.5 0.4]
Mean :  [0.1 0.4 0.8]
Covariance :
[[3 0 0]
 [0 3 0]
 [0 0 3]]
Prior :  0.25

Result of DF on passing x :  -5.826028393736074
```

C) Two were written, one for uni and one for multi since I couldn't figure out how to get it to recognize whether the input was a single value or an array.

Code:

```
def EuclideanU( x, y ):
    return sqrt((x-y)**2)

def EuclideanM( x, y ):
    return sqrt(sum((x-y)**2))
```

Two points (3,0) and (0,3) were given. The result is show below.

Output:

```
F:\CollegeStuff\Sem2\PR\Assg1>python Extra.py
Distance between (3,0) and (0,3) is 4.242640687119285
```

D) One new numpy function is used here called np.matmul, which returns the result of a matrix multiplication between two operands. It's used a lot throughout the rest of the exercises when calculating the DFs.

Code:

```
def Mahalanobis( x, mean, covariance ):
    covInv = np.linalg.pinv(covariance)
    return sqrt(np.matmul(np.matmul( (x-mean).T, covInv), (x-mean)))

x = np.array([0.3,0.5,0.4])
mean = np.array([0.1,0.4,0.8])
covariance = np.array([ [3, 0, 0], [0, 3, 0], [0, 0, 3]])
prior = 1/4

print("Input vector x :", x)
print("Mean : ", mean)
print("Covariance :")
print(covariance)
print("Distance between x and the mean : ", Mahalanobis(x,mean,covariance))
```

Output:

```
F:\CollegeStuff\Sem2\PR\Assg1>python Extra.py
Input vector x : [0.3 0.5 0.4]
Mean :  [0.1 0.4 0.8]
Covariance :
[[3 0 0]
 [0 3 0]
 [0 0 3]]
Distance between x and the mean :  0.264575131106459
```

---

2. For the dichotomizers, the input vector x has to be passed to the function. For E and F, the priors have to be passed along with the input vector as well. The code for the dichotomizer and the one to display the parameters and plot the assumed distributions have been placed together in this report.
The UVDF and MVDF functions are from 1(B).

The test samples were read from a file into three arrays called samplesClassX, where X = 1,2,3, one for each class. That code isn't specified here. Each array contains ten samples.

The Empirical Training Error output for **all three** dichotomizers are shown in (B).

A) Two classes, one feature, equal priors

Code:

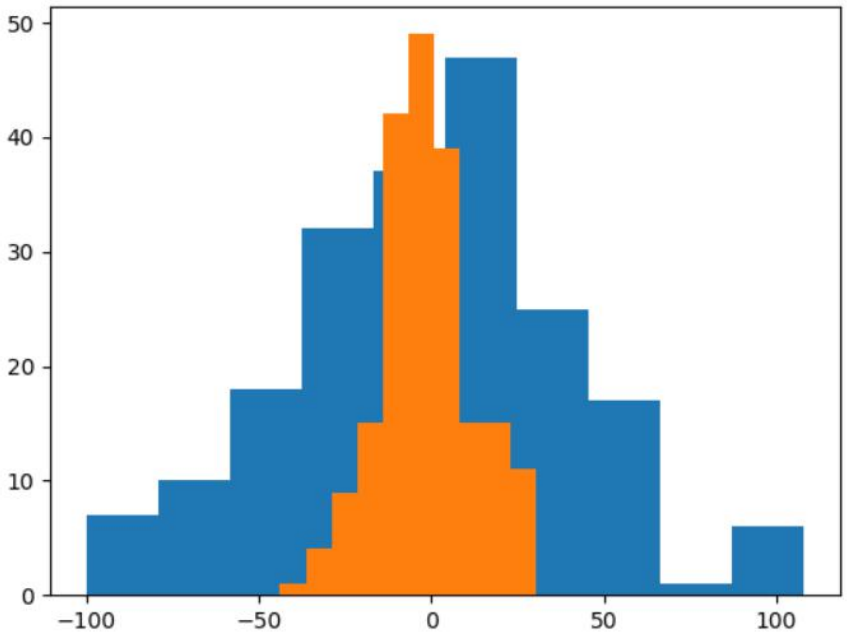
```
def FirstUVDichotomizer( x ):
    Pw1 = Pw2 = 0.5
```

```
        mu1 = np.array(sum(samplesClass1[:, 0])/len(samplesClass1))
        mu2 = np.array(sum(samplesClass2[:, 0])/len(samplesClass2))
        var1 = np.cov(samplesClass1[:, 0])
        var2 = np.cov(samplesClass2[:, 0])
        DichoResult = UVDF(x, mu1, var1, Pw1) - UVDF(x, mu2, var2, Pw2)
        return DichoResult

    def PlotDUV():
        mu1 = np.array(sum(samplesClass1[:, 0])/len(samplesClass1))
        mu2 = np.array(sum(samplesClass2[:, 0])/len(samplesClass2))
        var1 = np.cov(samplesClass1[:, 0])
        var2 = np.cov(samplesClass2[:, 0])
        print("Mean and Variance of w1: ", mu1, var1)
        print("Mean and Variance of w2: ", mu2, var2 )
        S = [ UVGaussianSample(mu1, var1) for i in range(200) ]
        R = [ UVGaussianSample(mu2, var2) for i in range(200) ]
        plt.hist(R)
        plt.hist(S)
        plt.show()

    #Test classification
    x = np.array(-5.5)
    ret = FirstUVDichotomizer(x)
    if ret > 0:   print(x, " belongs to class w1")
    else: print(x, "belongs to class w2")
    PlotDUV()
```

Output:

```
F:\CollegeStuff\Sem2\PR\Assg1>python Two.py
First Univariate Dichotomizer
Mean and Variance of w1:  -0.43999999999999984 ,  14.38051111111111
Mean and Variance of w2:  -0.543 ,  36.82933444444445
-5.5  belongs to class w1
```

Mean and Variance, and a test sample



Assumed Distributions of the first two classes, one feature histogram

---

B) Calculating Empirical Training Error for the three dichotomizers, percentage of mis-classifications were calculated out of the total number of samples in the set, which is twenty (for two classes).

The classification was done by checking the return value of the dichotomizer. The points from each of the samples in the data set were fed into the dichotomizer. If it belonged to the first class, the return value would've been positive and for the second class it would've been negative. Misclassifications would mean that the opposite happens, which is what was checked.

The total count of misclassifications is divided by twenty and then multiplied by a hundred to get the ETE.

Code:

```
    countUV = countMV1 = countMV2 = 0
    for i in range(0,10):
        ret = FirstUVDichotomizer(samplesClass1[i, 0])
        if ret < 0 : countUV +=1
        ret = FirstUVDichotomizer(samplesClass2[i, 0])
        if ret >= 0: countUV +=1

        ret = FirstMVDichotomizer(samplesClass1[i, 0:2])
        if ret < 0 : countMV1 +=1
        ret = FirstMVDichotomizer(samplesClass2[i, 0:2])
        if ret >= 0: countMV1 +=1

        ret = SecondMVDichotomizer(samplesClass1[i])
        if ret < 0 : countMV2 +=1
        ret = SecondMVDichotomizer(samplesClass2[i])
        if ret >= 0: countMV2 +=1

    print("Empirical Training Error for Univariate (One Feature) Dichotomizer: ", countUV/20*100, "%")
    print("Empirical Training Error for Multivariate (Two Features) Dichotomizer: ", countMV1/20*100,
"%")
    print("Empirical Training Error for Multivariate (All Features) Dichotomizer: ", countMV2/20*100,
"%")
```

Output:

```
F:\CollegeStuff\Sem2\PR\Assg1>python Two.py
Empirical Training Error for Univariate (One Feature) Dichotomizer:  50.0 %
Empirical Training Error for Multivariate (Two Features) Dichotomizer:  40.0 %
Empirical Training Error for Multivariate (All Features) Dichotomizer:  15.0 %
```

C)Two classes, first two features, equal priors

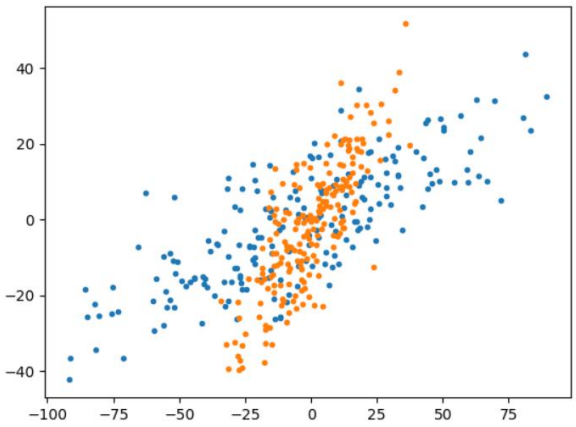
Code:

```
def FirstMVDichotomizer( x ):
    Pw1 = Pw2 = 0.5
    mu1 = np.array(sum(samplesClass1[:, 0:2])/len(samplesClass1))
    mu2 = np.array(sum(samplesClass2[:, 0:2])/len(samplesClass2))
    var1 = np.cov(samplesClass1[:, 0:2].T)
    var2 = np.cov(samplesClass2[:, 0:2].T)
    DichoResult = MVDF(x, mu1, var1, Pw1) - MVDF(x, mu2, var2, Pw2)
    return DichoResult

def PlotDMVI():
    mu1 = np.array(sum(samplesClass1[:, 0:2])/len(samplesClass1))
    mu2 = np.array(sum(samplesClass2[:, 0:2])/len(samplesClass2))
    var1 = np.cov(samplesClass1[:, 0:2].T)
    var2 = np.cov(samplesClass2[:, 0:2].T)
    print("First Multivariate Dichotomizer")
    print("Mean and Variance of w1: ", mu1)
    print(var1)
    print("Mean and Variance of w2: ", mu2, ", ")
    print(var2)
    S = np.array([ MVGaussianSample(mu1, var1, 2) for i in range(200) ])
    R = np.array([ MVGaussianSample(mu2, var2, 2) for i in range(200) ])
    plt.plot(R[:,0], R[:,1], '.')
    plt.plot(S[:,0], S[:,1], '.')
    plt.show()

x = [0,1.6]
ret = FirstMVDichotomizer(x)
if ret > 0:   print(x, " belongs to class w1")
else: print(x, "belongs to class w2")
PlotDMVI()
```

Output:

```
F:\CollegeStuff\Sem2\PR\Assg1>python Two.py
First Multivariate Dichotomizer
Mean and Variance of w1:  [-0.44  -1.749]
[[14.38051111  7.69537778]
 [ 7.69537778 14.62312111]]
Mean and Variance of w2:  [-0.543 -0.816] ,
[[36.82933444  9.87034667]
 [ 9.87034667 13.35347111]]
[0, 1.6]  belongs to class w1
```

Mean and Variance, and a test sample



Assumed Distributions of the first two classes, two features scatter plot

---

D)Two classes, all three features, equal priors

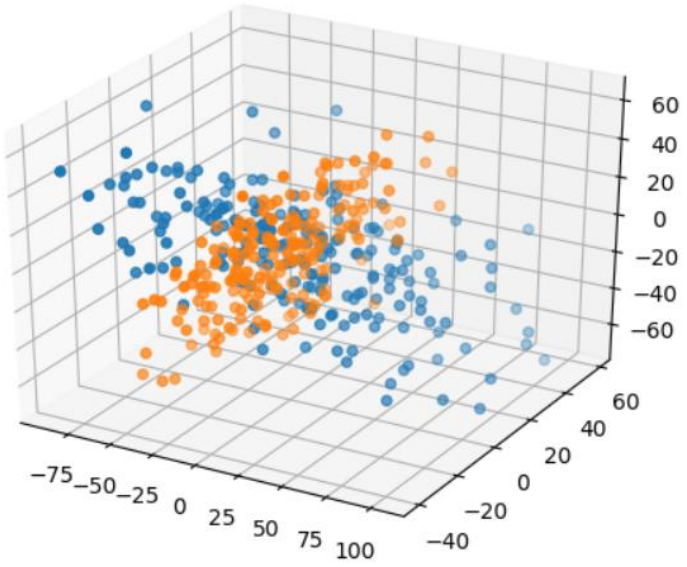
Code:

```
def SecondMVDichotomizer( x ):
    Pw1 = Pw2 = 0.5
    mu1 = np.array(sum(samplesClass1)/len(samplesClass1))
    mu2 = np.array(sum(samplesClass2)/len(samplesClass2))
    var1 = np.cov(samplesClass1.T)
    var2 = np.cov(samplesClass2.T)
    DichoResult = MVDF(x, mu1, var1, Pw1) - MVDF(x, mu2, var2, Pw2)
    return DichoResult

def PlotDMVII():
    mu1 = np.array(sum(samplesClass1)/len(samplesClass1))
    mu2 = np.array(sum(samplesClass2)/len(samplesClass2))
    var1 = np.cov(samplesClass1.T)
    var2 = np.cov(samplesClass2.T)
    print("Second Multivariate Dichotomizer")
    print("Mean and Variance of w1: ", mu1 )
    print(var1)
    print("Mean and Variance of w2: ", mu2 )
    print(var2)
    S = np.array([ MVGaussianSample(mu1, var1, 3) for i in range(200) ])
    R = np.array([ MVGaussianSample(mu2, var2, 3) for i in range(200) ])
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.scatter(R[:,0], R[:,1], R[:, 2])
    ax.scatter(S[:,0], S[:,1], S[:, 2])
    plt.show()

x = [-1,3.4,1]
ret = SecondMVDichotomizer(x)
if ret > 0:   print(x, " belongs to class w1")
else: print(x, "belongs to class w2")
PlotDMVII()
```

Output:

```
F:\CollegeStuff\Sem2\PR\Assg1>python Two.py
Second Multivariate Dichotomizer
Mean and Variance of w1:  [-0.44  -1.749 -0.766]
[[14.38051111  7.69537778  4.12232222]
 [ 7.69537778 14.62312111  3.90684   ]
 [ 4.12232222  3.90684    19.72453778]]
Mean and Variance of w2:  [-0.543 -0.816 -0.542]
[[ 36.82933444   9.87034667 -16.36675111]
 [  9.87034667  13.35347111   0.58833111]
 [-16.36675111   0.58833111  18.42121778]]
[-1, 3.4, 1] belongs to class w2
```

Mean and Variance, and a test sample



Assumed Distributions of the first two classes, three features scatter plot

---

E)Three classes, all three features, equal priors. Given set of test points are (1,2,1), (5,3,2), (0,0,0) and (1,0,0).

Code:

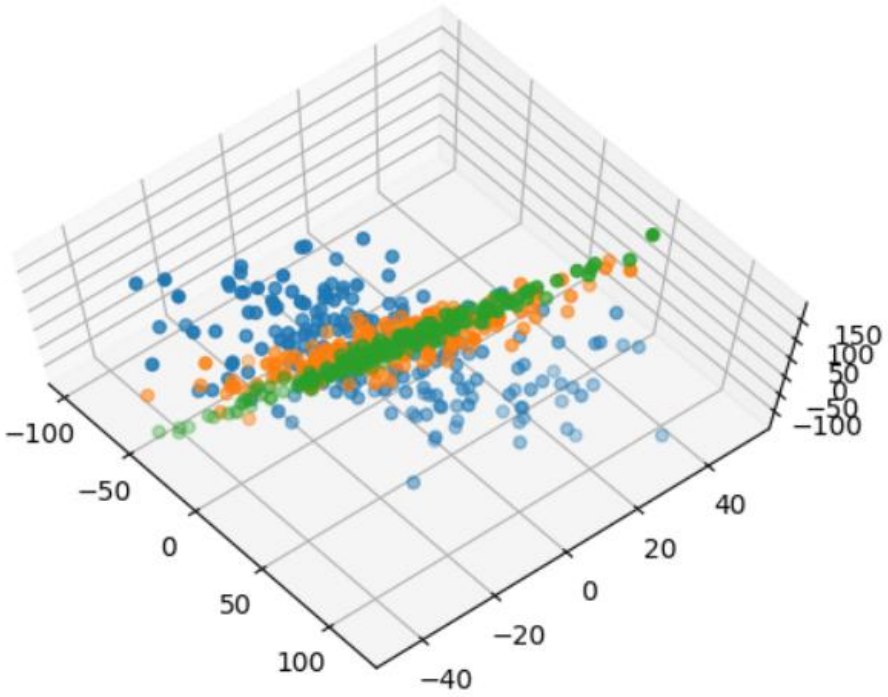
```
def ThirdMVDF( x, Pw1, Pw2, Pw3 ):
    mu1 = np.array(sum(samplesClass1)/len(samplesClass1))
    mu2 = np.array(sum(samplesClass2)/len(samplesClass2))
    mu3 = np.array(sum(samplesClass3)/len(samplesClass3))
    var1 = np.cov(samplesClass1.T)
    var2 = np.cov(samplesClass2.T)
    var3 = np.cov(samplesClass3.T)

    results = []
    results.append((MVDF(x, mu1, var1, Pw1), "w1"))
    results.append((MVDF(x, mu2, var2, Pw2), "w2"))
    results.append((MVDF(x, mu3, var3, Pw3), "w3"))

    A = max(results)
    print(x, " belongs to class ", A[1])
```

```
def PlotDMVIII():
    mu1 = np.array(sum(samplesClass1)/len(samplesClass1))
    mu2 = np.array(sum(samplesClass2)/len(samplesClass2))
    mu3 = np.array(sum(samplesClass3)/len(samplesClass3))
    var1 = np.cov(samplesClass1.T)
    var2 = np.cov(samplesClass2.T)
    var3 = np.cov(samplesClass3.T)
    print("Classifier for all three classes")
    print("Mean and Variance of w1: ", mu1)
    print(var1)
    print("Mean and Variance of w2: ", mu2)
    print(var2)
    print("Mean and Variance of w3: ", mu3)
    print(var3)
    S = np.array([ MVGaussianSample(mu1, var1, 3) for i in range(200) ])
    R = np.array([ MVGaussianSample(mu2, var2, 3) for i in range(200) ])
    T = np.array([ MVGaussianSample(mu3, var3, 3) for i in range(200) ])
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.scatter(R[:,0], R[:,1], R[:, 2])
    ax.scatter(S[:,0], S[:,1], S[:, 2])
    ax.scatter(T[:,0], T[:,1], T[:, 2])
    plt.show()


PlotDMVIII()
print("Classifying given four points with priors (0.33, 0.33, 0.33)")
ThirdMVDF([1,2,1], 0.33, 0.33, 0.33)
ThirdMVDF([5,3,2], 0.33, 0.33, 0.33)
ThirdMVDF([0,0,0], 0.33, 0.33, 0.33)
ThirdMVDF([1,0,0], 0.33, 0.33, 0.33)
```

Output:

```
F:\CollegeStuff\Sem2\PR\Assg1>python Two.py
Classifier for all three classes
Mean and Variance of w1:  [-0.44  -1.749 -0.766]
[[14.38051111  7.69537778  4.12232222]
 [ 7.69537778 14.62312111  3.90684   ]
 [ 4.12232222  3.90684    19.72453778]]

Mean and Variance of w2:  [-0.543 -0.816 -0.542]
[[ 36.82933444   9.87034667 -16.36675111]
 [  9.87034667  13.35347111   0.58833111]
 [-16.36675111   0.58833111  18.42121778]]

Mean and Variance of w3:  [3.883 1.376 1.58 ]
[[ 8.30475667  7.44494667 13.14957778]
 [ 7.44494667  8.56044889 11.60861111]
 [13.14957778 11.60861111 47.28728889]]

Classifying given four points with priors (0.33, 0.33, 0.33)
[1, 2, 1]  belongs to class  w2
[5, 3, 2]  belongs to class  w3
[0, 0, 0]  belongs to class  w1
[1, 0, 0]  belongs to class  w1
```

Mean and Variance, classifying the given points



Assumed Distributions of all three classes, three features scatter plot

---

E) All classes, all features, $P(\omega_1) = 0.8$, $P(\omega_2) = P(\omega_3) = 0.1$
Only the driver code changes compared to (E)

Code:

```
print("Classifying given four points with priors (0.8, 0.1, 0.1)")
ThirdMVDF([1,2,1], 0.8, 0.1, 0.1)
ThirdMVDF([5,3,2], 0.8, 0.1, 0.1)
ThirdMVDF([0,0,0], 0.8, 0.1, 0.1)
ThirdMVDF([1,0,0], 0.8, 0.1, 0.1)
```

Output:

```
F:\CollegeStuff\Sem2\PR\Assg1>python Two.py
Classifying given four points with priors (0.8, 0.1, 0.1)
[1, 2, 1]  belongs to class  w1
[5, 3, 2]  belongs to class  w1
[0, 0, 0]  belongs to class  w1
[1, 0, 0]  belongs to class  w1
```

Classifying the given points, with the new priors

---

3. The input data set was read into three arrays called *setosa, versicolor* and *virginica.* Each contains 50 samples.
Three classes were written, one for each case of the Discriminant Function. Three objects were made for each. When an object is instantiated, the array containing the samples of the class is passed. The mean and covariance for the class will be calculated. The method Result can then be used to get the value of the DF for some input vector x. The changeCov or changeVar can be used to change the covariance or variance of the class. They are used in case II and case I respectively.

The Result method uses the equations given in the Anaylsis document.
The ETE was calculated to see how the three cases perform on the given data set.
The distributions of the three classes were also plotted.

Code for the Classes:

```
#Case I Linear DF
class LDFI:
    def __init__(self, X):
        self.computeMuSigma(X)

    def computeMuSigma(self, X):
        self.mean = np.array(sum(X)/len(X))
        self.cov = np.cov(X.T)
        self.var = 1

    def changeVar(self, var):
        self.var = var

    def Result(self, x):
        A = np.matmul(self.mean.T, x)/(self.var)**2
        B = -0.5*np.matmul(self.mean.T, self.mean)/(self.var)**2 + np.log(0.33)
        return A+B


#Case II Linear DF
class LDF:
    def __init__(self, X):
        self.computeMuSigma(X)

    def computeMuSigma(self, X):
        self.mean = np.array(sum(X)/len(X))
        self.cov = np.cov(X.T)
        self.covInv = np.linalg.pinv(self.cov)
        self.covDet = np.linalg.det(self.cov)

    def changeCov(self, cov):
        self.cov = cov

    def Result(self, x):
        A = np.matmul(np.matmul(self.covInv, self.mean).T, x)
        B = -0.5*np.matmul(np.matmul(self.mean.T, self.covInv), self.mean) + log(0.33)
        return A+B


#Quadratic DF
class QDF:
    def __init__(self, X):
        self.computeMuSigma(X)

    def computeMuSigma(self, X):
        self.mean = np.array(sum(X)/len(X))
        self.cov = np.cov(X.T)
```

```
            self.covInv = np.linalg.pinv(self.cov)
            self.covDet = np.linalg.det(self.cov)

        def Result(self, x):
            A = -0.5*np.matmul(np.matmul(x.T, self.covInv), x)
            B = np.matmul(np.matmul(self.covInv, self.mean), x)
            C = -0.5*np.matmul(np.matmul(self.mean.T, self.covInv), self.mean) + log(0.33)
-0.5*log(self.covDet)
            return A + B + C
```

Instantiating the objects and setting up the average covariance/variance:

```
#Linear Discriminant Functions for each class (CASE I)
SetosaLDFI = LDFI(setosa)
VersicLDFI = LDFI(versicolor)
VirginLDFI = LDFI(virginica)

#Linear Discriminant Functions for each class (CASE II)
SetosaLDF = LDF(setosa)
VersicLDF = LDF(versicolor)
VirginLDF = LDF(virginica)

#Quadratic Discriminant Functions for each class
SetosaQDF = QDF(setosa)
VersicQDF = QDF(versicolor)
VirginQDF = QDF(virginica)


#Computing average covariance of the three classes
avgCov = (SetosaLDF.cov + VersicLDF.cov + VirginLDF.cov)/3
for i in range(0,4):
    for j in range(0,4):
        avgCov[j][i] = (avgCov[i][j]+avgCov[j][i])/2          #Making it symmetric

#Here is where it's set to CASE II
SetosaLDF.changeCov(avgCov)
VersicLDF.changeCov(avgCov)
VirginLDF.changeCov(avgCov)


#Computing average variance of the average covariance for case I
avgCov2 = avgCov * np.eye(4,4)
var = (np.mean(avgCov2[[0,1,2,3],[0,1,2,3]]))

#Here is where it's set to CASE I
SetosaLDFI.changeVar(var)
VersicLDFI.changeVar(var)
VirginLDFI.changeVar(var)
```

Calculating ETE:

```
countLDF = countLDFI = countQDF = 0

#Calculating ETE for QDF based classifier
for i in range(0, 50):
    A = SetosaQDF.Result(virginica[i])
    B = VersicQDF.Result(virginica[i])
    C = VirginQDF.Result(virginica[i])
    if C < A or C < B :
        countQDF += 1

    A = SetosaQDF.Result(setosa[i])
    B = VersicQDF.Result(setosa[i])
    C = VirginQDF.Result(setosa[i])
    if A < B or A < C :
```

```
            countQDF += 1

        A = SetosaQDF.Result(versicolor[i])
        B = VersicQDF.Result(versicolor[i])
        C = VirginQDF.Result(versicolor[i])
        if B < A or B < C :
            countQDF += 1

    #Calculating ETE for LDFI based classifier
    for i in range(0, 50):
        A = SetosaLDFI.Result(virginica[i])
        B = VersicLDFI.Result(virginica[i])
        C = VirginLDFI.Result(virginica[i])
        if C < A or C < B :
            countLDFI += 1

        A = SetosaLDFI.Result(setosa[i])
        B = VersicLDFI.Result(setosa[i])
        C = VirginLDFI.Result(setosa[i])
        if A < B or A < C :
            countLDFI += 1

        A = SetosaLDFI.Result(versicolor[i])
        B = VersicLDFI.Result(versicolor[i])
        C = VirginLDFI.Result(versicolor[i])
        if B < A or B < C :
            countLDFI += 1

    #Calculating ETE for LDF based classifier
    for i in range(0, 50):
        A = SetosaLDF.Result(virginica[i])
        B = VersicLDF.Result(virginica[i])
        C = VirginLDF.Result(virginica[i])
        if C < A or C < B :
            countLDF += 1

        A = SetosaLDF.Result(setosa[i])
        B = VersicLDF.Result(setosa[i])
        C = VirginLDF.Result(setosa[i])
        if A < B or A < C :
            countLDF += 1

        A = SetosaLDF.Result(versicolor[i])
        B = VersicLDF.Result(versicolor[i])
        C = VirginLDF.Result(versicolor[i])
        if B < A or B < C :
            countLDF += 1

    print("\nEmprical Training Error for LDF (Case I) : ", (countLDFI/150)*100, "%")
    print("Emprical Training Error for LDF (Case II) : ", (countLDF/150)*100, "%")
    print("Emprical Training Error for QDF : ", (countQDF/150)*100, "%")
```

Displaying information on the classes:

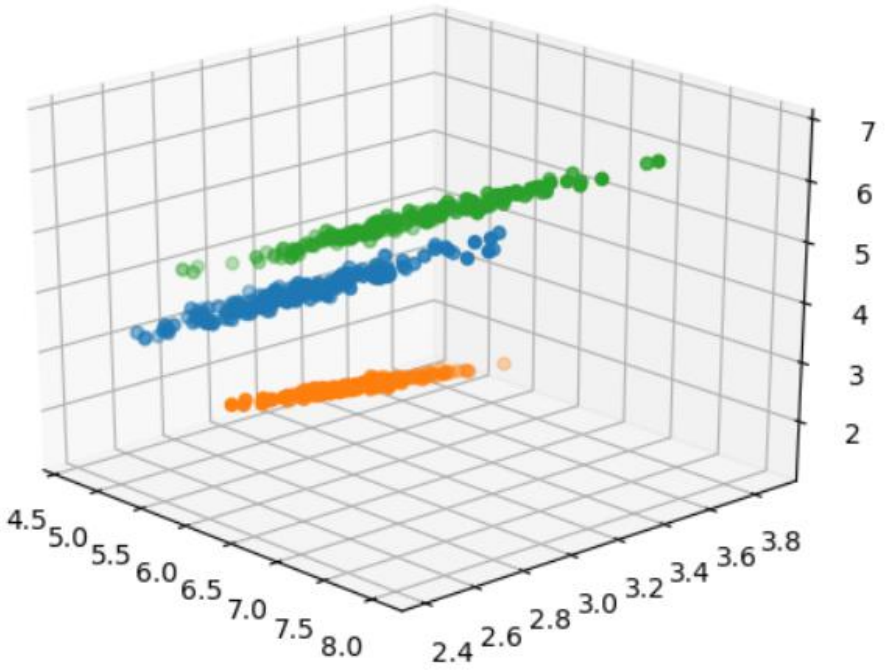
```
    print("For the Setosa Class,")
    print("Mean : ", SetosaQDF.mean)
    print("Covariance : ")
    print(SetosaQDF.cov, "\n")

    print("For the Versicolor Class,")
    print("Mean : ", VersicQDF.mean)
    print("Covariance : ")
    print(VersicQDF.cov, "\n")

    print("For the Virginica Class,")
```

```
print("Mean : ", VirginQDF.mean)
print("Covariance : ")
print(VirginQDF.cov, "\n")

print("Average Covariance for Case II was taken as")
print(avgCov)

print("\nAverage Variance for Case I was taken as")
print(var)

S = np.array([ MVGaussianSample(SetosaQDF.mean, SetosaQDF.cov, 4) for i in range(200) ])
R = np.array([ MVGaussianSample(VersicQDF.mean, VersicQDF.cov, 4) for i in range(200) ])
T = np.array([ MVGaussianSample(VirginQDF.mean, VirginQDF.cov, 4) for i in range(200) ])
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(R[:,0], R[:,1], R[:, 2])
ax.scatter(S[:,0], S[:,1], S[:, 2])
ax.scatter(T[:,0], T[:,1], T[:, 2])
plt.show()
```

Output:

```
F:\CollegeStuff\Sem2\PR\Assg1>python Three.py
For the Setosa Class,
Mean :  [5.006 3.418 1.464 0.244]
Covariance :
[[0.12424898 0.10029796 0.01613878 0.01054694]
 [0.10029796 0.14517959 0.01168163 0.01143673]
 [0.01613878 0.01168163 0.03010612 0.00569796]
 [0.01054694 0.01143673 0.00569796 0.01149388]]

For the Versicolor Class,
Mean :  [5.936 2.77  4.26  1.326]
Covariance :
[[0.26643265 0.08518367 0.18289796 0.05577959]
 [0.08518367 0.09846939 0.08265306 0.04120408]
 [0.18289796 0.08265306 0.22081633 0.07310204]
 [0.05577959 0.04120408 0.07310204 0.03910612]]

For the Virginica Class,
Mean :  [6.588 2.974 5.552 2.026]
Covariance :
[[0.40434286 0.09376327 0.3032898  0.04909388]
 [0.09376327 0.10400408 0.07137959 0.04762857]
 [0.3032898  0.07137959 0.30458776 0.04882449]
 [0.04909388 0.04762857 0.04882449 0.07543265]]

Average Covariance for Case II was taken as
[[0.26500816 0.09308163 0.16744218 0.03847347]
 [0.09308163 0.11588435 0.0552381  0.03342313]
 [0.16744218 0.0552381  0.18517007 0.0425415 ]
 [0.03847347 0.03342313 0.0425415  0.04201088]]

Average Variance for Case I was taken as
0.15201836734693877

Emprical Training Error for LDF (Case I) :  7.333333333333333 %
Emprical Training Error for LDF (Case II) :  66.66666666666666 %
Emprical Training Error for QDF :  2.0 %
```

Mean and Variance of the three classes, the average covariance/variance and the ETEs



Assumed distributions of the three Iris classes with the first three features (sepal length, width, petal length)