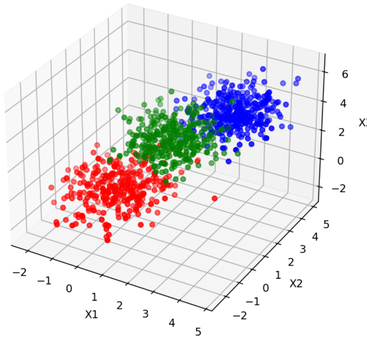
Question 2.6

In [1]:
```
# Loading of relevant libraries
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

import seaborn as sns
```

In [2]:
```
N = 1000

np.random.seed(4)

Pw0 = Pw1 = Pw2 = 1/3       # class a priori probabilities are equal

m0 = np.array([0, 0, 0])
m1 = np.array([1, 2, 2])
m2 = np.array([3, 3, 4])


S1 = np.array([[0.8,0.2,0.1],
               [0.2,0.8,0.2],
               [0.1,0.2,0.8]])
S2 = np.array([[0.6,0.01,0.01],
               [0.01,0.8,0.01],
               [0.01,0.01,0.6]])
S3 = np.array([[0.6,0.1,0.1],
               [0.1,0.6,0.1],
               [0.1,0.1,0.6]])
```

In [3]:
```
## training set

Xtr_w0 = np.random.multivariate_normal(m0, S1, 333)  # vectors for class_0
ytr_w0 = 0*np.ones((333, 1))                         # labels for class_0

Xtr_w1 = np.random.multivariate_normal(m1, S2, 333)  # vectors for class_1
ytr_w1 = 1*np.ones((333, 1))                         # labels for class_1

Xtr_w2 = np.random.multivariate_normal(m2, S3, 333)  # vectors for class_2
ytr_w2 = 2*np.ones((333, 1))                         # labels for class_2

# collection in a single set for data and labels

Xtr = np.concatenate((Xtr_w0, Xtr_w1, Xtr_w2), axis = 0)
ytr = np.concatenate((ytr_w0, ytr_w1, ytr_w2), axis = 0)



## test set

Xte_w0 = np.random.multivariate_normal(m0, S1, 333)    # vectors for class_0
yte_w0 = 0*np.ones((333, 1))                           # labels for class_0

Xte_w1 = np.random.multivariate_normal(m1, S2, 333)    # vectors for class_1
yte_w1 = 1*np.ones((333, 1))                           # labels for class_1

Xte_w2 = np.random.multivariate_normal(m2, S3, 333)    # vectors for class_2
yte_w2 = 2*np.ones((333, 1))                           # labels for class_2

# collection in a single set for data and labels

Xte = np.concatenate((Xte_w0, Xte_w1, Xte_w2), axis = 0)
yte = np.concatenate((yte_w0, yte_w1, yte_w2), axis = 0)
```

In [4]:
```
# data ploting

%matplotlib notebook
fig = plt.figure(figsize = (6,6))
ax  = fig.add_subplot(projection = "3d")

ax.scatter(Xtr_w0[:,0], Xtr_w0[:,1], Xtr_w0[:,2], marker = "o", color = "r", label = "Class 0")
ax.scatter(Xtr_w1[:,0], Xtr_w1[:,1], Xtr_w1[:,2], marker = "o", color = "g", label = "Class 1")
ax.scatter(Xtr_w2[:,0], Xtr_w2[:,1], Xtr_w2[:,2], marker = "o", color = "b", label = "Class 2")
ax.set_xlabel('X1')
ax.set_ylabel('X2')
ax.set_zlabel('X3')

plt.show()
```



In [5]:
```
# question -1 : Calculation of ML estimates

m0_hat = (1.0/(N/3))*np.sum(Xtr_w0, axis = 0)
S0_hat = (1.0/(N/3))*np.dot((Xtr_w0-m0_hat).T,(Xtr_w0-m0_hat))

m1_hat = (1.0/(N/3))*np.sum(Xtr_w1, axis = 0)
S1_hat = (1.0/(N/3))*np.dot((Xtr_w1-m1_hat).T,(Xtr_w1-m1_hat))

m2_hat = (1.0/(N/3))*np.sum(Xtr_w2, axis = 0)
S2_hat = (1.0/(N/3))*np.dot((Xtr_w2-m2_hat).T,(Xtr_w2-m2_hat))

S_hat = (1.0/3.0)*(S0_hat + S1_hat + S2_hat)
```

In [6]:
```
# Question - 2
# Mahalanobis distance calculation on the test set from the estimate mean of each class

inv_S = np.linalg.inv(S_hat)
dm_0 = np.sqrt(np.sum(np.dot((Xte-m0_hat), inv_S)*(Xte-m0_hat), axis = 1))
dm_1 = np.sqrt(np.sum(np.dot((Xte-m1_hat), inv_S)*(Xte-m1_hat), axis = 1))
dm_2 = np.sqrt(np.sum(np.dot((Xte-m2_hat), inv_S)*(Xte-m2_hat), axis = 1))

# Classification based on the calculated euclidean distances

dm_matrix = np.stack((dm_0, dm_1, dm_2), axis = 1)
Mahal_result = np.argmin(dm_matrix, axis = 1)
```

In [7]:
```
#Question - 3
def multivariate_normal_pdf_v2(x, mean, sigma):
    l = x.shape[1]
    det_S = np.linalg.det(sigma)
    norm_const = 1.0/((2.0*np.pi)**(l/2.0)*np.sqrt(det_S))
    inv_S = np.linalg.inv(sigma)
    a1    = np.sum(np.dot(x-mean, inv_S)*(x-mean), axis = 1)

    return norm_const*np.exp(-0.5*a1)
```

In [8]:
```
baydis_x1 = Pw0*multivariate_normal_pdf_v2(Xte, m0_hat,S_hat)

baydis_x2 = Pw1*multivariate_normal_pdf_v2(Xte, m1_hat,S_hat)


baydis_x3 = Pw2*multivariate_normal_pdf_v2(Xte, m2_hat,S_hat)


de_matrix = np.stack((baydis_x1, baydis_x2, baydis_x3), axis = 1)
Bayes_result = np.argmax(de_matrix, axis = 1)
```

In [9]:
```
# Question - 4
# To compute the error probability the classification results are compare with the reference matrix


#error_bayesian clasifier

error_bayesian = 1-np.sum(Bayes_result == yte.flatten())/N

#error_mahalanobis

error_mahalanobis = 1-np.sum(Mahal_result  == yte.flatten())/N

#print(error_bayesian)

print(error_bayesian)

#print(error_euclidean)

print(error_mahalanobis)
```

```
0.05900000000000005
0.05900000000000005
```

it is possible to observe that the error using both methods is the same due to the same probability that each class have