Homework 5

In [1]:
```
import numpy as np

Xp = np.array([
        [4, 2, 2, 3, 4, 6, 3, 8],
        [1, 4, 3, 6, 4, 2, 2, 3],
        [0, 1, 1, 0, -1, 0, 1, 0]
        ])

Xn = np.array([
        [9, 6, 9, 8, 10],
        [10, 8, 5, 7, 8],
        [1, 0, 0, 1, -1]
        ])
```

1. write code to compute the class specific means of data matrices Xp and Xn. Your code needs to return the mean as a {\bf column} vector.
2. use your code, compute the mean of matrices Xp and Xn given in the problem setting.

In [2]:
```
Xp_mean = np.mean(Xp, axis=1)
Xn_mean = np.mean(Xn, axis=1)
```

In [3]:
```
print('Xp_mean: ', Xp_mean)
print('Xn_mean: ', Xn_mean)
```

```
Xp_mean:  [4.    3.125 0.25 ]
Xn_mean:  [8.4 7.6 0.2]
```

1. write code to compute the class specific covariance matrices of data matrices Xp and Xn.
2. use your code, compute the class specific covariance matrices of data matrices Xp and Xn given in the problem setting.

In [4]:
```
Xp1 = np.zeros( (len(Xp), len(Xp[0]) ))
for i in range(len(Xp[0])):
        Xp1[:,i] = Xp[:,i] - Xp_mean

Xn1 = np.zeros( (len(Xn), len(Xn[0]) ))
for i in range(len(Xn[0])):
        Xn1[:,i] = Xn[:,i] - Xn_mean

Xp_Cov = Xp1.dot(Xp1.T)
Xn_Cov = Xn1.dot(Xn1.T)
```

In [5]:
```
print('Xp_Cov: ', Xp_Cov)
print('Xn_Cov: ', Xn_Cov)
```

```
Xp_Cov:  [[30.     -6.     -5.    ]
 [-6.     16.875 -1.25  ]
 [-5.     -1.25    3.5   ]]
Xn_Cov:  [[ 9.2 -0.2 -1.4]
 [-0.2 13.2  1.4]
 [-1.4  1.4  2.8]]
```

Covariance Matrix

$$C = XX^T = \sum_{k=1}^{n} (X_k - mean)(X_k - mean)^T$$

1. write code to compute between class scattering matrix Sb.
2. use your code, compute Sb for data given in the problem setting.

In [6]:
```
tmp = np.append(Xp, Xn, axis=1)
total_Mean = np.mean(tmp, axis=1)
Sb_Xp = len(total_Mean)*((Xp_mean.reshape(len(Xp_mean),1) - total_Mean
.reshape(len(total_Mean),1)).dot((Xp_mean.reshape(len(Xp_mean),1) - to
tal_Mean.reshape(len(total_Mean),1)).T))
Sb_Xn = len(total_Mean)*((Xn_mean.reshape(len(Xn_mean),1) - total_Mean
.reshape(len(total_Mean),1)).dot((Xn_mean.reshape(len(Xn_mean),1) - to
tal_Mean.reshape(len(total_Mean),1)).T))
Sb = Sb_Xp + Sb_Xn
```

In [7]:
```
print('Sb_Xp: ', Sb_Xp)
print('Sb_Xn: ', Sb_Xn)
print('Sb: ', Sb)
```

```
Sb_Xp:  [[ 8.59171598e+00  8.73816568e+00 -9.76331361e-02]
 [ 8.73816568e+00  8.88711169e+00 -9.92973373e-02]
 [-9.76331361e-02 -9.92973373e-02  1.10946746e-03]]
Sb_Xn:  [[ 2.19947929e+01  2.23697041e+01 -2.49940828e-01]
 [ 2.23697041e+01  2.27510059e+01 -2.54201183e-01]
 [-2.49940828e-01 -2.54201183e-01  2.84023669e-03]]
Sb:  [[ 3.05865089e+01  3.11078698e+01 -3.47573964e-01]
 [ 3.11078698e+01  3.16381176e+01 -3.53498521e-01]
 [-3.47573964e-01 -3.53498521e-01  3.94970414e-03]]
```

Between class scattering matrix

$$S_b = \frac{1}{n} \sum (\mu_+ - \mu_-)(\mu_+ - \mu_-)^T$$

where mean of positive data

$$\mu_+ = \frac{1}{n_+} \sum_i x_i^+$$

mean of negative data

$$\mu_- = \frac{1}{n_-} \sum_i x_i^-$$

So, mean of real data (Xp and Xn)

$$\mu_+$$

mean of total data

$$\mu_-$$

In order to calculate Sb, array Xp and Xn are combined. And then, Sb_Xp and Sb_Xn are calculated respectively.

1. write code to compute the within class scattering matrix Sw.
2. use your code, compute Sw for data given in the problem setting.

In [8]:
```
Sw = Xp_Cov + Xn_Cov
```

In [9]:
```
print('Sw: ', Sw)
```

```
Sw:  [[39.2    -6.2    -6.4   ]
 [-6.2    30.075  0.15  ]
 [-6.4     0.15   6.3   ]]
```

Within class scattering matrix

$$S_w = \frac{n_+}{n_-} S_+ + \frac{n_+}{n_-} S_-$$

where covariance matrix of positive data

$$S_+ = \frac{1}{n_+} \sum_i (x_i^+ - \mu_+)(x_i^+ - \mu_+)^T$$

covariance matrix of negative data

$$S_+ = \frac{1}{n_-} \sum_i (x_i^+ - \mu_-)(x_i^+ - \mu_-)^T$$

So, we can use covariance matrix we already calculated above in order for Sw

1. write code to compute the LDA projection by solving the generalized eigenvalue decomposition problem. [use function numpy.linalg.eig]
2. use your code, compute the LDA projection for the data given in the problem setting. You should see that the second eigenvalue is zero.

In [10]:

```
eigenvalue, eigenvector = np.linalg.eig(np.linalg.inv(Sw).dot(Sb))

# Sorting eigenvalue and eigenvector from highest to lowest
arrNum = np.argsort(eigenvalue)[::-1]
eigenvector_sort = eigenvector[:,arrNum]
eigenvalue_sort = eigenvalue[arrNum]

v = eigenvector_sort[:, :1]

print("eigenvalue: ", eigenvalue_sort)
print("Second eigenvalue: ", eigenvalue_sort[1])
print("We can see that the second eigenvalue is almost zero like given instruction")
```

```
eigenvalue:  [2.43615153e+00 3.14701096e-16 4.05475959e-18]
Second eigenvalue:  3.1470109648107917e-16
We can see that the second eigenvalue is almost zero like given inst
ruction
```

3 eigenvalues are 2.43615153e+00 3.14701096e-16 4.05475959e-18. Like a given instruction, we can see that the second eigenvalue is almost zero.

Generalized eigenvalue problem. Let's solve

$$\lambda S_w v = S_b v$$

When Sw is invertible v is eigenvector of the top eigenvalue for matrix

$$S_w^{-1} S_b$$

with

$$\lambda$$

being the corresponding eigenvalue.

Therefore, in order to get eigenvalue and eigenvector, we use

$$\lambda S_w v = S_b v$$

1. collect all previous steps, write a function with name mybLDA_train to perform binary LDA, which takes inputs of two data matrices Xp and Xn assuming column data, and return the optimal LDA projection direction as a unit vector.

In [11]:
```
def mybLDA_train(Xp, Xn):
        Xp_mean = np.mean(Xp, axis=1)
        Xn_mean = np.mean(Xn, axis=1)

        Xp1 = np.zeros( (len(Xp), len(Xp[0]) ))
        for i in range(len(Xp[0])):
                Xp1[:,i] = Xp[:,i] - Xp_mean

        Xn1 = np.zeros( (len(Xn), len(Xn[0]) ))
        for i in range(len(Xn[0])):
                Xn1[:,i] = Xn[:,i] - Xn_mean

        Xp_Cov = Xp1.dot(Xp1.T)
        Xn_Cov = Xn1.dot(Xn1.T)

        tmp = np.append(Xp, Xn, axis=1)
        total_Mean = np.mean(tmp, axis=1)
        Sb_Xp = len(total_Mean)*((Xp_mean.reshape(len(Xp_mean),1) - to
tal_Mean.reshape(len(total_Mean),1)).dot((Xp_mean.reshape(len(Xp_mean)
,1) - total_Mean.reshape(len(total_Mean),1)).T))
        Sb_Xn = len(total_Mean)*((Xn_mean.reshape(len(Xn_mean),1) - to
tal_Mean.reshape(len(total_Mean),1)).dot((Xn_mean.reshape(len(Xn_mean)
,1) - total_Mean.reshape(len(total_Mean),1)).T))
        Sb = Sb_Xp + Sb_Xn

        Sw = Xp_Cov + Xn_Cov

        eigenvalue, eigenvector = np.linalg.eig(np.linalg.inv(Sw).dot(
Sb))

        arrNum = np.argsort(eigenvalue)[::-1]
        eigenvector_sort = eigenvector[:,arrNum]
        eigenvalue_sort = eigenvalue[arrNum]

        bestRep = eigenvalue_sort[0] / np.sum(eigenvalue)
        bestRepVector = eigenvector_sort[:,:1]
        # Choose best 1 eigenvectors
        projectionDirection = eigenvector_sort[:, :1]

        return projectionDirection
```

1. write a function with name mybLDA_classify which takes a data matrix X and a projection direction v, returns a row vector r that has size as the number of rows in X, and r_i =+1 if the ith column of X is from the class as in Xp, and r_i =-1 if the ith column in X is from the class as in Xn.
2. Run your function, mybLDA_train, on the data given in the problem setting, and then use the obtained projection direction and your function mybLDA_classify to classify the following data set X

In [12]:
```
X = np.array([
        [1.3, 2.4, 6.7, 2.2, 3.4, 3.2],
        [8.1, 7.6, 2.1, 1.1, 0.5, 7.4],
        [-1, 2, 3, 2, 0, 2]
        ])
v = eigenvector_sort[:, :1]
```

In [13]:
```
def mybLDA_classify(X, v):
        Xp_lda = (Xp.T).dot(v)
        Xn_lda = (Xn.T).dot(v)
        X_lda = (X.T).dot(v)
        mean = (np.mean(Xp_lda) + np.mean(Xn_lda))/2
        r = np.zeros((len(X_lda),1))

        for i in range(len(X_lda)):
                if X_lda[i] < mean:
                        r[i] = 1
                else:
                        r[i] = -1

        return r
```

Lets' transform the test data, which is array X, on the new subspace. Therefore,

$$Xv$$

To find threshold in order for which columns are from, I use mean value. So, I calculated total mean of Xp and Xn. A given array A and v, which is eigenvector, can be multiplied. This is called X_lda. If total mean of Xp and Xn is greater than X_lda, then r_i = +1, which means that the ith column of X is from the class as in Xp. Otherwise, r_i = -1, which indicates the ith column of X is from the class as in Xn.

Below is entier code for homework 5

In [14]:
```
import numpy as np


def getXp():
        Xp = np.array([
        [4, 2, 2, 3, 4, 6, 3, 8],
        [1, 4, 3, 6, 4, 2, 2, 3],
        [0, 1, 1, 0, -1, 0, 1, 0]
        ])

        return Xp

def getXn():
        Xn = np.array([
        [9, 6, 9, 8, 10],
        [10, 8, 5, 7, 8],
        [1, 0, 0, 1, -1]
```

```
        ])

        return Xn

def getX():
        X = np.array([
        [1.3, 2.4, 6.7, 2.2, 3.4, 3.2],
        [8.1, 7.6, 2.1, 1.1, 0.5, 7.4],
        [-1, 2, 3, 2, 0, 2]
        ])

        return X

def mybLDA_train(Xp, Xn):
        Xp_mean = np.mean(Xp, axis=1)
        Xn_mean = np.mean(Xn, axis=1)

        Xp1 = np.zeros( (len(Xp), len(Xp[0]) ))
        for i in range(len(Xp[0])):
                Xp1[:,i] = Xp[:,i] - Xp_mean

        Xn1 = np.zeros( (len(Xn), len(Xn[0]) ))
        for i in range(len(Xn[0])):
                Xn1[:,i] = Xn[:,i] - Xn_mean

        Xp_Cov = Xp1.dot(Xp1.T)
        Xn_Cov = Xn1.dot(Xn1.T)

        tmp = np.append(Xp, Xn, axis=1)
        total_Mean = np.mean(tmp, axis=1)
        Sb_Xp = len(total_Mean)*((Xp_mean.reshape(len(Xp_mean),1) - to
tal_Mean.reshape(len(total_Mean),1)).dot((Xp_mean.reshape(len(Xp_mean)
,1) - total_Mean.reshape(len(total_Mean),1)).T))
        Sb_Xn = len(total_Mean)*((Xn_mean.reshape(len(Xn_mean),1) - to
tal_Mean.reshape(len(total_Mean),1)).dot((Xn_mean.reshape(len(Xn_mean)
,1) - total_Mean.reshape(len(total_Mean),1)).T))
        Sb = Sb_Xp + Sb_Xn

        Sw = Xp_Cov + Xn_Cov

        eigenvalue, eigenvector = np.linalg.eig(np.linalg.inv(Sw).dot(
Sb))

        arrNum = np.argsort(eigenvalue)[::-1]
        eigenvector_sort = eigenvector[:,arrNum]
        eigenvalue_sort = eigenvalue[arrNum]

        bestRep = eigenvalue_sort[0] / np.sum(eigenvalue)
        bestRepVector = eigenvector_sort[:,:1]
        # Choose best 1 eigenvectors
        projectionDirection = eigenvector_sort[:, :1]

        return projectionDirection
```

```python
def mybLDA_classify(X, v):
        Xp_lda = (getXp().T).dot(v)
        Xn_lda = (getXn().T).dot(v)
        X_lda = (X.T).dot(v)
        mean = (np.mean(Xp_lda) + np.mean(Xn_lda))/2
        r = np.zeros((len(X_lda),1))

        for i in range(len(X_lda)):
                if X_lda[i] < mean:
                        r[i] = 1
                else:
                        r[i] = -1

        return r.T


def main():
        Xp = getXp()
        Xn = getXn()
        X = getX()
        v = mybLDA_train(Xp, Xn)
        r = mybLDA_classify(X, v)
        print(r)



if __name__ == "__main__":
        main()
```

```
[[ 1. -1.  1.  1.  1. -1.]]
```

In [ ]: