Assignment 6

2020年3月21日 星期六　　下午7:21

1. Power iteration

pick $\underline{B}_0$

for $k = 1, 2, \ldots$ to converge

$$\underline{B}_k = \underline{A}\,\underline{B}_{k-1} / \| \underline{A}\,\underline{B}_{k-1} \|_2$$

end

$$\underline{V}_1 = \underline{B}_{end}$$

$$AB_{k-1} = A^k B_0 = V \Lambda^k V^T$$

let $B_0 = \underline{V} g = \underline{V} \begin{bmatrix} g_1 \\ \vdots \\ g_m \end{bmatrix}$

$\lambda_i = \sigma_i^2$

then $A^k B_0 = V \Lambda^k g$

$$= \lambda_1^k g_1 \underline{V} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \left(\frac{\lambda_2}{\lambda_1}\right)^k & 0 \\ 0 & 0 & \left(\frac{\lambda_m}{\lambda_1}\right)^k \end{bmatrix} \begin{bmatrix} 1 \\ \frac{g_2}{g_1} \\ \vdots \\ \frac{g_m}{g_1} \end{bmatrix}$$

$k \to \infty \quad A^k B_0 \to \lambda_1^k g_1 \underline{V}_1$

$$B_k \to \frac{\lambda_1^k g_1 \underline{V}_1}{\| \lambda_1^k g_1 \underline{V}_1 \|_2} = \underline{V}_1$$

Since $\underline{A}$ is rank-1 $\quad \lambda_2 = \lambda_3 = \cdots \lambda_n = 0$

So just after one iteration, $AB_0 = \lambda_1 g_1 V_1$

$$B_1 = \frac{AB_0}{\|AB_0\|} = V_1$$

Actually, since A is rank 1 and symmetric.

$$A = \sigma_1 U_1 V_1^T$$

and $U_1 = V_1$

$$B_1 = \frac{AB_0}{\|AB_0\|_2} = \frac{\sigma_1 U_1 V_1^T B_0}{\sigma_1 \|U_1\| (V_1^T B_0)} = U_1$$

Therefore, number of iterations needed B

2. a) No. Not pass $\underline{0}$

b) substract mean of data

c) Yes

d) Fits well

e) $\bar{X}_2 = \sigma_1 U_1 V_1^T$

So $X_{2i} = \sigma_1 U_1 V_{1i} = a w_i$

So $w_i = \sigma_1 V_{1i}$ where V_{1i} is the i th element

of V_1

f) $\underline{b}$ is a vector whose value of each entry is mean value of [illegible] data

g) $\underline{E} = X - (\tilde{X}_2 + X_{AVE})$

$= X_2 - \sigma_1 u_1 v_1^T$

$= \sum_{i=2}^{3} \sigma_i u_i v_i^T$

$\|\underline{E}\|_F^2 = \| \sum_{i=2}^{3} \sigma_i u_i v_i^T \|_F^2 = \sigma_2^2 + \sigma_3^2$

h) see attached

i) $w_{2i} = \sigma_2 v_{2i}$

Display see attached

Yes. Yes.

j) $\|\underline{E}\|_F^2 = \| \sigma_3 u_3 v_3^T \|_F^2 = \sigma_3^2$

k) rank 1: 626.69

rank 2: 152.95

3. a) final estimate 10.94%

3. a) final estimate 10.94%

b) final estimate 10.16%

Details see code attaced

In [26]:

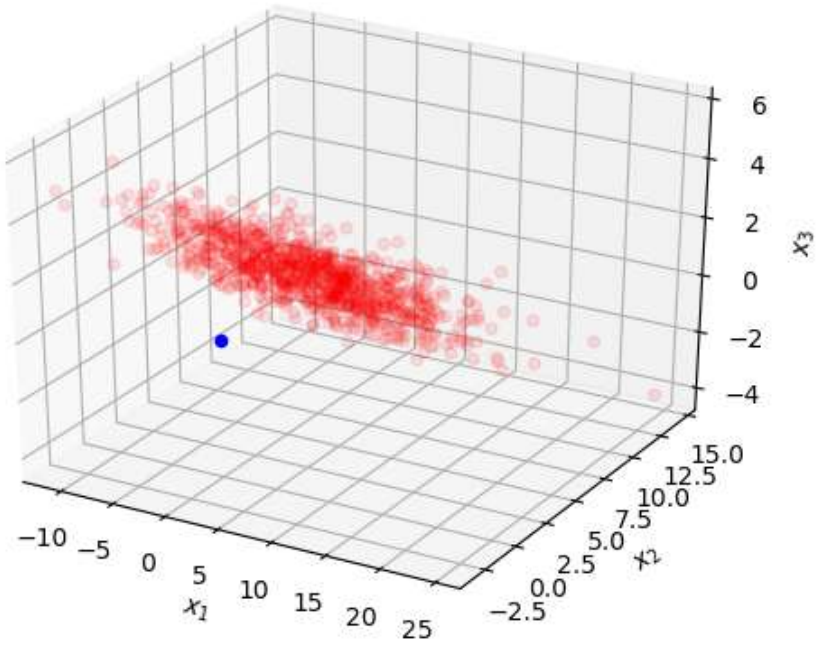
```
# Enable interactive rotation of graph
%matplotlib notebook

import numpy as np
from scipy.io import loadmat
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Load data for activity
X = np.loadtxt('sdata.csv',delimiter=',')
X.shape
```

Out[26]:

(1000, 3)

In [27]:

```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

ax.scatter(X[:,0], X[:,1], X[:,2], c='r', marker='o', alpha=0.1)
ax.scatter(0,0,0,c='b', marker='o')
ax.set_xlabel('$x_1$')
ax.set_ylabel('$x_2$')
ax.set_zlabel('$x_3$')

plt.show()
```



In [28]:

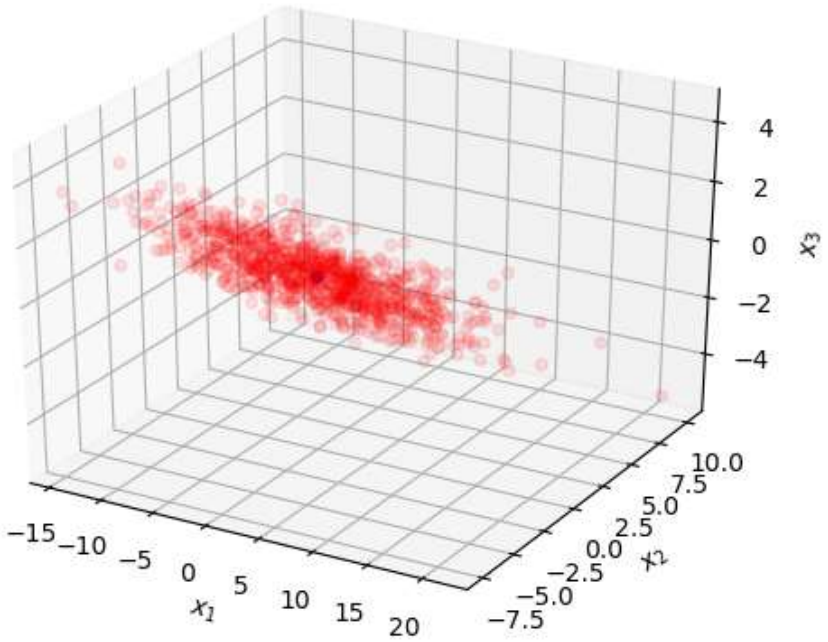
```
# Subtract mean
X_m = X - np.mean(X, 0)
np.mean(X, 0)
```

Out[28]:

```
array([3.17585563, 4.24319511, 1.25610797])
```

In [29]:

```
# display zero mean scatter plot
fig = plt.figure()

ax = fig.add_subplot(111, projection='3d')
ax.scatter(X_m[:,0], X_m[:,1], X_m[:,2], c='r', marker='o', alpha=0.1)

ax.scatter(0,0,0,c='b', marker='o')
ax.set_xlabel('$x_1$')
ax.set_ylabel('$x_2$')
ax.set_zlabel('$x_3$')

plt.show()
```

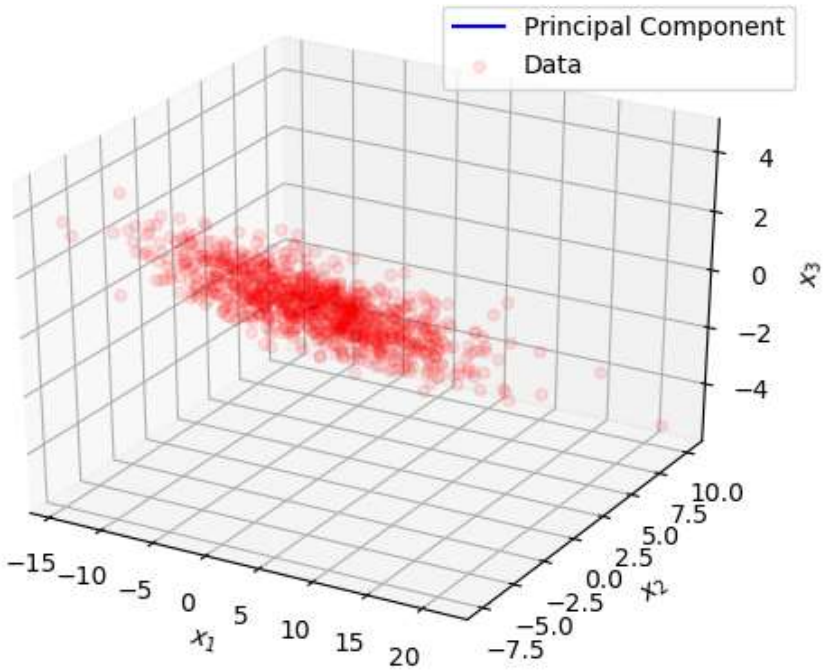


In [30]:

```
# Use SVD to find first principal component

U,s,VT = np.linalg.svd(X_m,full_matrices=False)

# complete the next line of code to assign the first principal component to a
a = U[:,0]
```

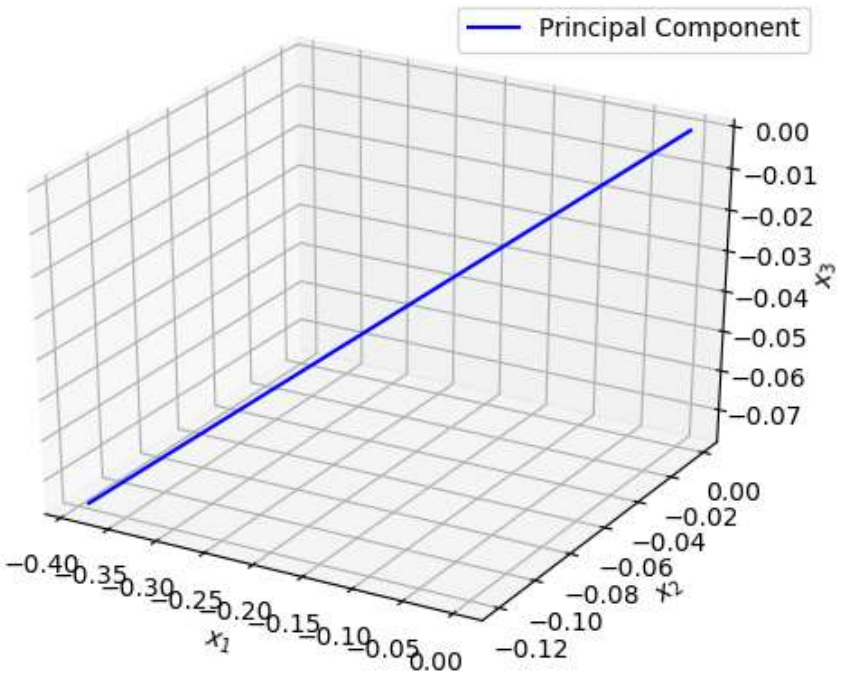
In [31]:

```
# display zero mean scatter plot and first principal component

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

#scale length of line by root mean square of data for display
ss = s[0]/np.sqrt(np.shape(X_m)[0])

ax.scatter(X_m[:,0], X_m[:,1], X_m[:,2], c='r', marker='o', label='Data', alpha=0.1)

ax.plot([0,ss*a[0]],[0,ss*a[1]],[0,ss*a[2]], c='b',label='Principal Component')
# ax.plot([0,a[0]],[0,a[1]],[0,a[2]], c='b',label='Principal Component')

ax.set_xlabel('$x_1$')
ax.set_ylabel('$x_2$')
ax.set_zlabel('$x_3$')


ax.legend()
plt.show()
```



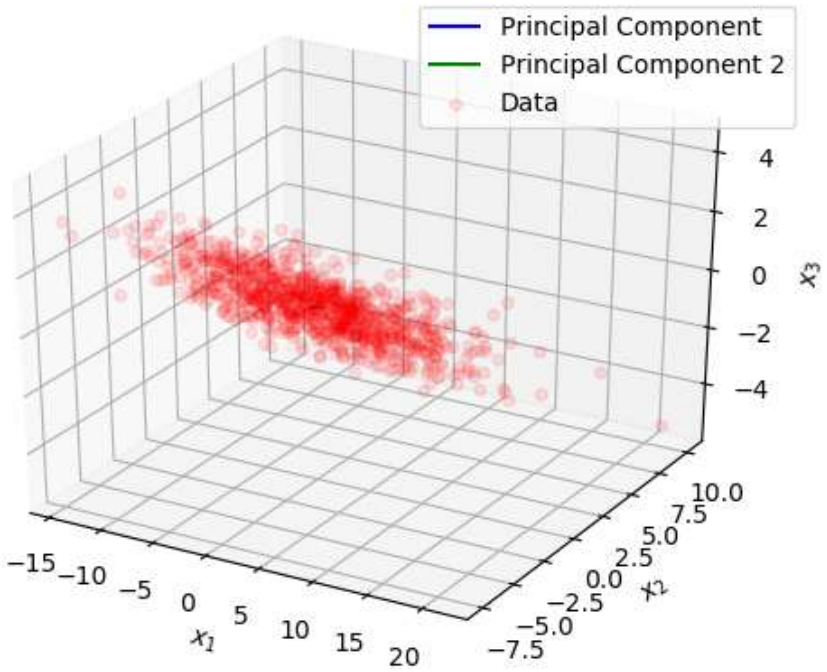
In [32]:

```
# display zero mean scatter plot and first principal component

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

#scale length of line by root mean square of data for display
ss = s[0]/np.sqrt(np.shape(X_m)[0])

# ax.scatter(X_m[:,0], X_m[:,1], X_m[:,2], c='r', marker='o', label='Data', alpha=0.1)

ax.plot([0,ss*a[0]],[0,ss*a[1]],[0,ss*a[2]], c='b',label='Principal Component')

ax.set_xlabel('$x_1$')
ax.set_ylabel('$x_2$')
ax.set_zlabel('$x_3$')


ax.legend()
plt.show()
```



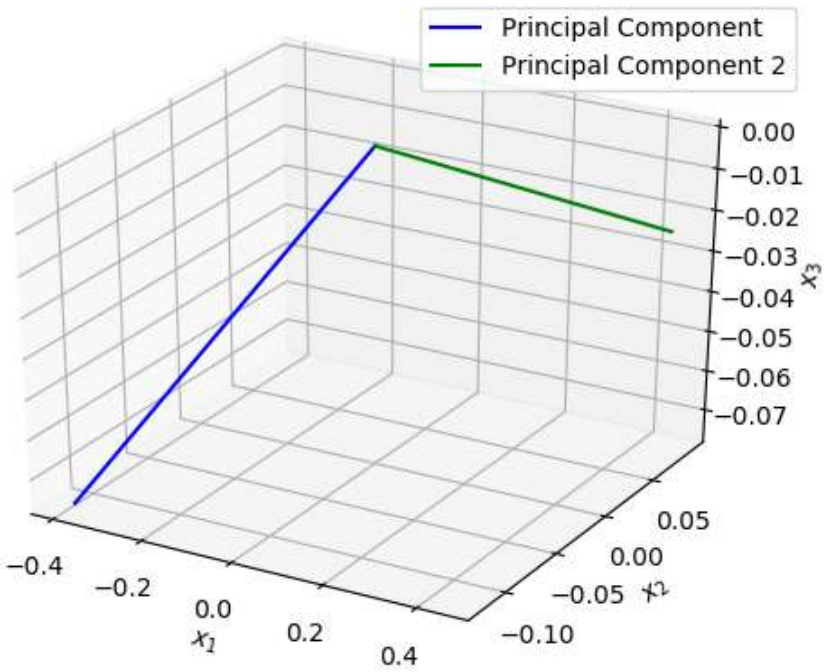
In [33]:

```
# display zero mean scatter plot and first two principal components
b = U[:,1]
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

#scale length of line by root mean square of data for display
ss = s[0]/np.sqrt(np.shape(X_m)[0])

ax.scatter(X_m[:,0], X_m[:,1], X_m[:,2], c='r', marker='o', label='Data', alpha=0.1)

ax.plot([0,ss*a[0]],[0,ss*a[1]],[0,ss*a[2]], c='b',label='Principal Component')
ax.plot([0,ss*b[0]],[0,ss*b[1]],[0,ss*b[2]], c='g',label='Principal Component 2')

ax.set_xlabel('$x_1$')
ax.set_ylabel('$x_2$')
ax.set_zlabel('$x_3$')


ax.legend()
plt.show()
```



In [34]:

```
# display zero mean scatter plot and first two principal components
b = U[:,1]
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

#scale length of line by root mean square of data for display
ss = s[0]/np.sqrt(np.shape(X_m)[0])

# ax.scatter(X_m[:,0], X_m[:,1], X_m[:,2], c='r', marker='o', label='Data', alpha=0.1)

ax.plot([0,ss*a[0]],[0,ss*a[1]],[0,ss*a[2]], c='b',label='Principal Component')
ax.plot([0,ss*b[0]],[0,ss*b[1]],[0,ss*b[2]], c='g',label='Principal Component 2')

ax.set_xlabel('$x_1$')
ax.set_ylabel('$x_2$')
ax.set_zlabel('$x_3$')


ax.legend()
plt.show()
```

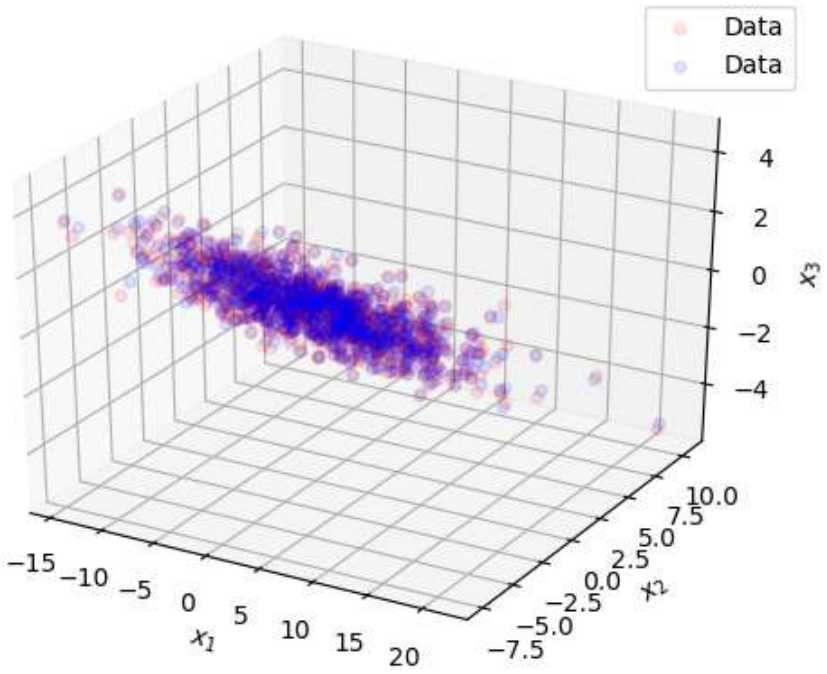


In [35]:

```
Ar = np.zeros((len(U), len(VT)))
for j in range(2):
        Ar += s[j] * np.outer(U.T[j], VT[j])
```

In [36]:

```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

#scale length of line by root mean square of data for display
ss = s[0]/np.sqrt(np.shape(X_m)[0])

ax.scatter(X_m[:,0], X_m[:,1], X_m[:,2], c='r', marker='o', label='Data', alpha=0.1)
ax.scatter(Ar[:,0], Ar[:,1], Ar[:,2], c='b', marker='o', label='Data', alpha=0.1)

ax.set_xlabel('$x_1$')
ax.set_ylabel('$x_2$')
ax.set_zlabel('$x_3$')


ax.legend()
plt.show()
```



In [40]:

```
# Ar = np.zeros((len(U), len(VT)))
# for j in range(1):
#         Ar += s[j] * np.outer(U.T[j], VT[j])
# Er = X_m - Ar
# print(Er)
# err_fro = np.linalg.norm(Er,ord='fro')
# print("rank1 err:{}".format(err_fro))
print(s[1]*s[1]+s[2]*s[2])
```

626.6899203862782

In [41]:

```
# Ar = np.zeros((len(U), len(VT)))
# for j in range(2):
#         Ar += s[j] * np.outer(U.T[j], VT[j])
# Er = X_m - Ar
# err_fro = np.linalg.norm(Er,ord='fro')
# print("rank2 err:{}".format(err_fro))
print(s[2]*s[2])
# print(Er)
```

152.9455757788646

In [2]:

```
import numpy as np
from scipy.io import loadmat
import matplotlib.pyplot as plt

in_data = loadmat('face_emotion_data.mat')
#loadmat() loads a matlab workspace into a python dictionary, where the names of the variables are i
#in the dictionary.  To see what variables are loaded, uncomment the line below:
print([key for key in in_data])

X = in_data['X']
y = in_data['y']
n = len(X)
print(n)
# print(len(X))
# print(128*9)
# print(X)
# plt.scatter(X,y)
# plt.title('data')
# plt.show()
```

```
['__header__', '__version__', '__globals__', 'y', 'X']
128
```

In [ ]:

```

```

In [16]:

```
err_list=[]
for i in range(7):
    for j in range(i+1, 8):
        # define cut-off indexes
        start_index_1 = i*16
        end_index_1 = (i+1)*16
        start_index_2 = j*16
        end_index_2 = (j+1)*16
        # extract training data
        X_train = np.vstack((X[0:start_index_1,:],X[end_index_1:start_index_2,:], X[end_index_2:-1,
        y_train = np.vstack((y[0:start_index_1,:],y[end_index_1:start_index_2,:], y[end_index_2:-1,
        # extract validation parameter data
        X_p=X[start_index_1:end_index_1,:]
        y_p=y[start_index_1:end_index_1,:]
        # extract validation data
        X_val=X[start_index_2:end_index_2,:]
        y_val=y[start_index_2:end_index_2,:]

        U,s,VT = np.linalg.svd(X_train,full_matrices=False)

        best_err = 1
        best_w = None
        best_r = None
        for r_0 in range(9):
            r = r_0 + 1
            ind=int(r-1)
            s_inverse = np.empty(len(s))
            for k in range(len(s)):
                s_inverse[k] = 1.0 / s[k]
            w = VT.T[:,0:ind]@np.diag(s_inverse[0:ind])@U.T[0:ind,:]@y_train
            y_test = np.sign(X_p@w)
            err = np.mean(y_p!=y_test)
            if err < best_err:
                best_err = err
                best_w = w
                best_r = r
        y_predict = np.sign(X_val@best_w)
        err_list.append(np.mean(y_val!=y_predict))

        # exchange p and val
        X_p=X[start_index_2:end_index_2,:]
        y_p=y[start_index_2:end_index_2,:]
        X_val=X[start_index_1:end_index_1,:]
        y_val=y[start_index_1:end_index_1,:]

        best_err = 1
        best_w = None
        best_r = None
        for r_0 in range(9):
            r = r_0 + 1
            ind=int(r-1)
            s_inverse = np.empty(len(s))
            for k in range(len(s)):
                s_inverse[k] = 1.0 / s[k]
#             print(np.diag(s_inverse))
#             print(np.linalg.inv(np.diag(s)))
#             print(np.diag(s)**-1)
            w = VT.T[:,0:ind]@np.diag(s_inverse[0:ind])@U.T[0:ind,:]@y_train
            y_test = np.sign(X_p@w)
```

```
            err = np.mean(y_p!=y_test)
            if err < best_err:
                best_err = err
                best_w = w
                best_r = r
        y_predict = np.sign(X_val@best_w)
        err_list.append(np.mean(y_val!=y_predict))

avg_err=np.mean(err_list)
print("3a Average error rate is {}%".format(avg_err*100))
```

```
3a Average error rate is 10.9375%
```

In [6]:

```
err_list=[]
r_vals = np.array([0,1/2,1,2,4,8,16])
for i in range(7):
    for j in range(i+1, 8):
        # define cut-off indexes
        start_index_1 = i*16
        end_index_1 = (i+1)*16
        start_index_2 = j*16
        end_index_2 = (j+1)*16
        # extract training data
        X_train = np.vstack((X[0:start_index_1,:],X[end_index_1:start_index_2,:], X[end_index_2:-1,
        y_train = np.vstack((y[0:start_index_1,:],y[end_index_1:start_index_2,:], y[end_index_2:-1,
        # extract validation parameter data
        X_p=X[start_index_1:end_index_1,:]
        y_p=y[start_index_1:end_index_1,:]
        # extract validation data
        X_val=X[start_index_2:end_index_2,:]
        y_val=y[start_index_2:end_index_2,:]

        U,s,VT = np.linalg.svd(X_train,full_matrices=False)

        best_err = 1
        best_w = None
        best_r = None
        for r in r_vals:
            D = np.linalg.inv(np.diag(s)@np.diag(s) + r*np.identity(len(s)))
            w = VT.T@D@U.T@y_train
            y_test = np.sign(X_p@w)
            err = np.mean(y_p!=y_test)
            if err < best_err:
                best_err = err
                best_w = w
                best_r = r
        y_predict = np.sign(X_val@best_w)
        err_list.append(np.mean(y_val!=y_predict))

        # exchange p and val
        X_p=X[start_index_2:end_index_2,:]
        y_p=y[start_index_2:end_index_2,:]
        X_val=X[start_index_1:end_index_1,:]
        y_val=y[start_index_1:end_index_1,:]

        best_err = 1
        best_w = None
        best_r = None
        for r in r_vals:
            D = np.linalg.inv(np.diag(s)@np.diag(s) + r*np.identity(len(s)))
            w = VT.T@D@U.T@y_train
            y_test = np.sign(X_p@w)
            err = np.mean(y_p!=y_test)
            if err < best_err:
                best_err = err
                best_w = w
                best_r = r
        y_predict = np.sign(X_val@best_w)
        err_list.append(np.mean(y_val!=y_predict))

# print(len(err_list))
avg_err=np.mean(err_list)
```

```
print("3b Average error rate is {}%".format(avg_err*100))
```

```
3b Average error rate is 10.15625%
```

In [ ]:

```
```