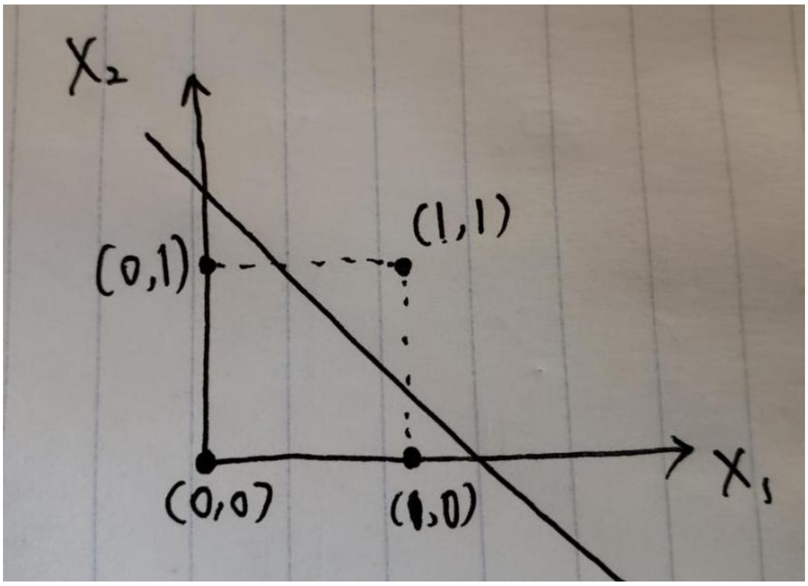
# JichenDai_HW#2

# 10448922

**1.**

**(1) Yes, It is linearly separable.**

As we can see in the graph below, there is a line separate positive and negative.




**(2) Yes, it is possible.**

Set Learning Rate = 0.2.

Since x1 + x2 - 1/2 = 0,

w0 = -0.5, w1 = 1, w2 = 1

**Input P1:**

y = 0*1 + 1*1 - 0.5

= 0.5 > 0

Prediction is **wrong**, update the weights:

w0 = w0*(1+0.2) = 0.6

w1 = w1*(1-0.2) = 0.8

w2 = w2*(1-0.2) = 0.8

**Input P2:**

y = 1*0.8 + 1*0.8 - 0.6

= 1 > 0

Prediction is **correct**

**Input P3:**

y = 1*0.8 + 0*0.8 - 0.6

= 0.2 > 0

Prediction is **wrong**, update the weights:

$w_0 = w_0 * (1+0.2) = 0.72$
$w_1 = w_1 * (1-0.2) = 0.64$
$w_2 = w_2 * (1-0.2) = 0.64$

**Input P4:**
$y = 0*0.64 + 0*0.64 - 0.72$
$= -0.72 < 0$
Prediction is **correct**.

**Input P1:**
$y = 0*0.64 + 1*0.64 - 0.72$
$= -0.08 < 0$
Prediction is **correct**.

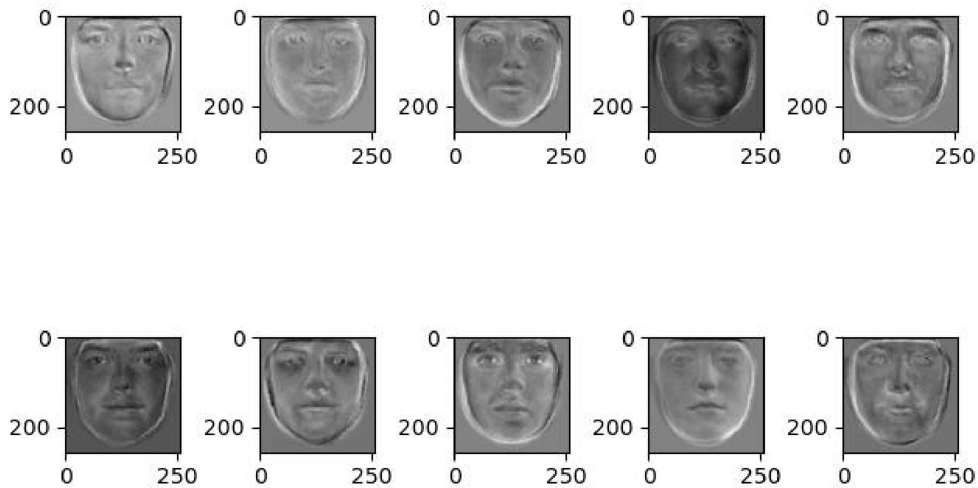
**Input P2:**
$y = 1*0.64 + 1*0.64 - 0.72$
$= 0.56 > 0$
Prediction is **correct**.

**Input P3:**
$y = 1*0.64 + 0*0.64 - 0.72$
$= -0.08 < 0$
Prediction is **correct**.

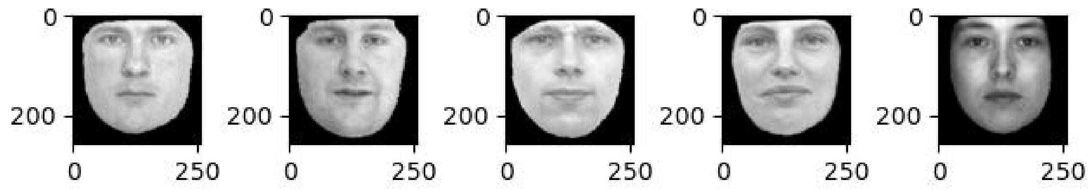
**Input P4:**
$y = 0*0.64 + 0*0.64 - 0.72$
$= -0.72 < 0$
Prediction is **correct**.

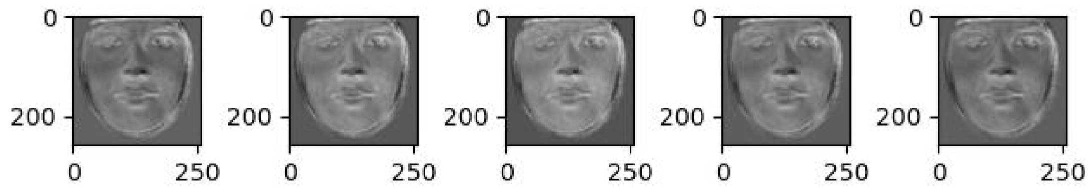
So, now the weight can correctly classify all 4 points, final decision boundary is:
**0.64** * X1 + **0.64** * X2 **- 0.72** = 0

**2**
**(1)**

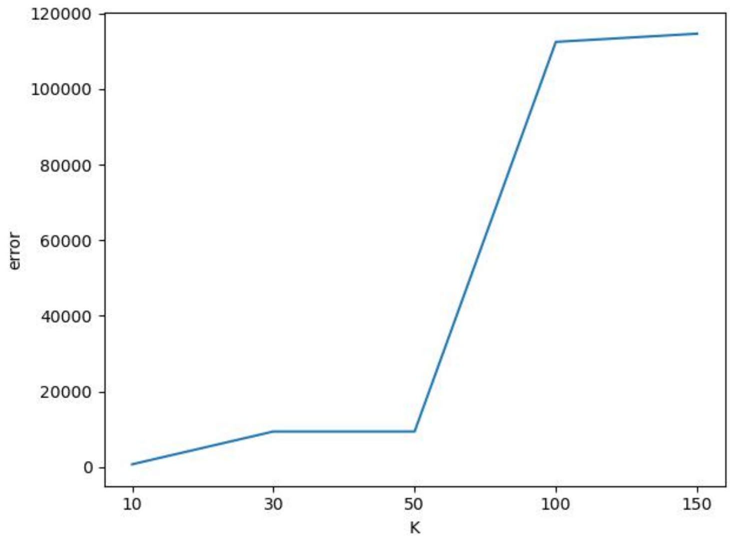


**(2) error is 9441**




**(3)**




**Source code:**

```
import numpy as np
from PIL import Image
from os import listdir
import matplotlib.pyplot as plt


# load data with shape 177*65536
def load_image(path):
    vectors = []
    for bmp in listdir(path):
        vector = np.array(Image.open("face_data/" + bmp)).flatten()
```

```
        vectors.append(vector)
    return np.array(vectors, dtype=np.float64)


def plot(images, row, col):
    plt.figure()
    for i in range(col * row):
        plt.subplot(row, col, i + 1)
        image = images[i].reshape(256,256)
        plt.imshow(image, cmap = 'gray')
    plt.show()


def cal_error(origin, recon):
    row, col = origin.shape
    sum = 0
    for i in range(origin.shape[0]):
        for j in range(col):
           sum += (origin[i,j] - recon[i,j])**2
    return sum / (row*col)


def norm(faces):
    norm = []
    for face in faces:
        if(face.sum() != 0):
            norm.append(face / face.sum())
        else:
            norm.append(face)
    return np.array(norm)




####################
# load data
data = load_image("face_data")
training, test = data[:156], data[157:]

# calculate mean (Y = X - avg(X))=>(156*65536)
[r, c] = training.shape
mean = np.mean(training, 0)
Y = training - mean

# compute covariance (Y * Y_t / size_train)=>(156*156)
cov = np.dot(Y, Y.T) / c
```

```python
# get eigen vetor
values, vectors = np.linalg.eig(cov)

# sort vectors according to values
ascending_order = np.argsort(values)
descending_order = ascending_order[::-1]
vectors = vectors[:, descending_order]

# select top (K = 30)=>(30*156)
selected_V = vectors[:30]

# do projection (Y_t * eigenvectors) =>(30*65536)
eigenfaces = np.dot(selected_V, Y)
eigenfaces = norm(eigenfaces)

# display top 10 eigenfaces
plot(eigenfaces[:10], 2, 5)


#################### Question 2 ######################
# show 5 original testing images
plot(test[:5], 1, 5)

# show 5 reconstructed ones.
Y_test = np.dot(test[:5] - mean, eigenfaces.T)
Y_test = np.dot(Y_test, eigenfaces) + mean
plot(Y_test, 1, 5)

# compute error value
error = cal_error(test[:5], Y_test)
print(error)


################ Question 3 #######################
def given_K_return_error(K):
    eigenfaces = np.dot(vectors[:K], Y)
    eigenfaces = norm(eigenfaces)

    Y_test = np.dot(test[:5] - mean, eigenfaces.T)
    Y_test = np.dot(Y_test, eigenfaces) + mean

    error = cal_error(test[:5], Y_test)
    return error
```

```python
Ks = [10, 20, 50, 100, 150]
errors = []

for K in Ks:
    errors.append(given_K_return_error(K))

plt.plot(['10', '30', '50', '100', '150'], errors)
plt.xlabel("K")
plt.ylabel("error")
plt.show()
```