CS559-B HW2

Name: Yiqun Peng

Problem 1:

(1) Yes, the positive and negative classes are linearly separable, because there can be a line that separates the points (1,1) from the points (0,0) (1,0) (0,1).

(2) Supposed $\eta = 1$, $Xb = 1$.
And if $f(x) <= 0$, $f(x) = 0$, else $f(x) = 1$.

xb	x1	x2	y
1	0	1	0
1	1	1	1
1	1	0	0
1	0	0	0

According to $x1 + x2 - 1/2 = 0$, $w = (wb, w1, w2) = (-1/2, 1, 1)$.

STEP 1:

① $x = (1, 0, 1)$, $w = (-1/2, 1, 1)$, $y = 0$

$f(x) = -1/2 + 0 + 1 = 1/2 > 0$, $f(x) = 1$

$\triangle w = \eta * (y - f(x)) * x = (-1, 0, -1)$

② $x = (1, 1, 1)$, $w = (-1/2, 1, 1)$, $y = 1$

$f(x) = -1/2 + 1 + 1 = 3/2 > 0$, $f(x) = 1$

$\triangle w = \eta * (y - f(x)) * x = (0, 0, 0)$

③ $x = (1, 1, 0)$, $w = (-1/2, 1, 1)$, $y = 0$

$f(x) = -1/2 + 1 + 0 = 1/2 > 0$, $f(x) = 1$

$\triangle w = \eta * (y - f(x)) * x = (-1, -1, 0)$

④ $x = (1, 0, 0)$, $w = (-1/2, 1, 1)$, $y = 0$

$f(x) = -1/2 + 0 + 0 = -1/2 < 0$, $f(x) = 0$

$\triangle w = \eta * (y - f(x)) * x = (0, 0, 0)$

$SUM(\triangle w) = (-2,-1,-1)$, $w = (-5/2, 0, 0)$

STEP 2:

① x = (1, 0, 1), w = (-5/2, 0, 0), y = 0

f(x) = -5/2 + 0 + 0 = -5/2 < 0, f(x) = 0

△w = η * (y – f(x)) * x = (0, 0, 0)

② x = (1, 1, 1), w = (-5/2, 0, 0), y = 1

f(x) = -5/2 + 0 + 0 = -5/2 < 0, f(x) = 0

△w = η * (y – f(x)) * x = (1, 1, 1)

③ x = (1, 1, 0), w = (-5/2, 0, 0), y = 0

f(x) = -5/2 + 0 + 0 = -5/2 < 0, f(x) = 0

△w = η * (y – f(x)) * x = (0, 0, 0)

④ x = (1, 0, 0), w = (-5/2, 0, 0), y = 0

f(x) = -5/2 + 0 + 0 = -5/2 < 0, f(x) = 0

△w = η * (y – f(x)) * x = (0, 0, 0)

SUM(△w) = (1, 1, 1), w = (-3/2, 1, 1)

STEP 3:

① x = (1, 0, 1), w = (-3/2, 1, 1), y = 0

f(x) = -3/2 + 0 + 1 = -1/2 < 0, f(x) = 0

△w = η * (y – f(x)) * x = (0, 0, 0)

② x = (1, 1, 1), w = (-3/2, 1, 1), y = 1

f(x) = -3/2 + 1 + 1 = 1/2 > 0, f(x) = 1

△w = η * (y – f(x)) * x = (0, 0, 0)

③ x = (1, 1, 0), w = (-3/2, 1, 1), y = 0

f(x) = -3/2 + 1 + 0 = -1/2 < 0, f(x) = 0

△w = η * (y – f(x)) * x = (0, 0, 0)

④ x = (1, 0, 0), w = (-3/2, 1, 1), y = 0

f(x) = -3/2 + 0 + 0 = -3/2 < 0, f(x) = 0

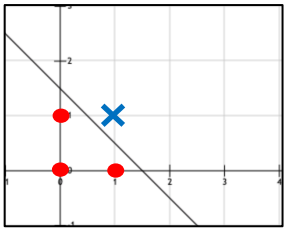
$\triangle w = \eta * (y - f(x)) * x = (0, 0, 0)$

$SUM(\triangle w) = (0, 0, 0)$

STOP

$\therefore$ $w = (-3/2, 1, 1)$

$\therefore$ Boundary: $-3/2+X1+X2 = 0$



Problem 2:

(1) PCA Code:

```
def doPCA(dataMat, K):
    N, D = dataMat.shape
    mean = np.mean(dataMat, axis=0)
    minus_mean = dataMat - mean
    covMultiplyX = np.dot(minus_mean,np.transpose(minus_mean)) * (1 / N)
    eigenVal, eigenVectMultiplyX = np.linalg.eig(covMultiplyX)
    eigenVect = np.dot(np.transpose(minus_mean), eigenVectMultiplyX)
    #print(eigenVal.shape, eigenVectMultiplyX.shape, eigenVect.shape)
    eigenValRank = np.argsort(eigenVal)
    eigenValRank = eigenValRank[:-(K+1):-1]
    # print("val",eigenVal.size,"vec",eigenVect.shape)
    eigenVect_norm = eigenVect / np.linalg.norm(eigenVect, axis=0)
    eigenVectByK = eigenVect[:, eigenValRank]
    eigenVectByK_norm = eigenVect_norm[:, eigenValRank]

    return eigenVectByK, eigenVectByK_norm, mean
```

Use OpenCV to read, process, and visualize images. Numpy is used to calculate the average face, obtain the eigenvalues and eigenvectors and unitization.

Eigenfaces (K = 30):

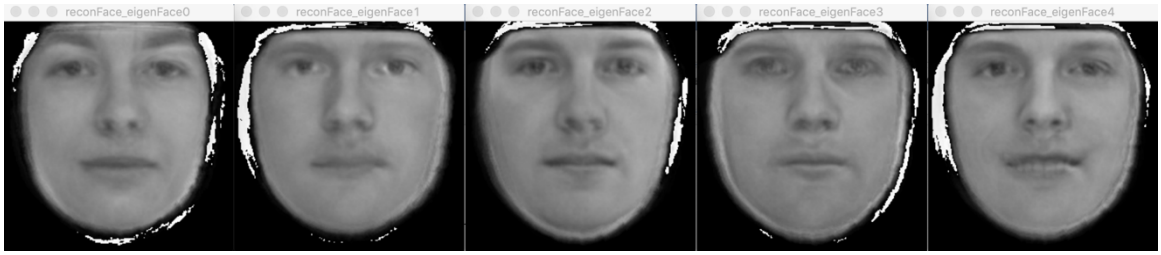
From the top left to the bottom right are the 10 eigen faces corresponding to the largest 10 eigen values.

(2) Reconstruction Code:

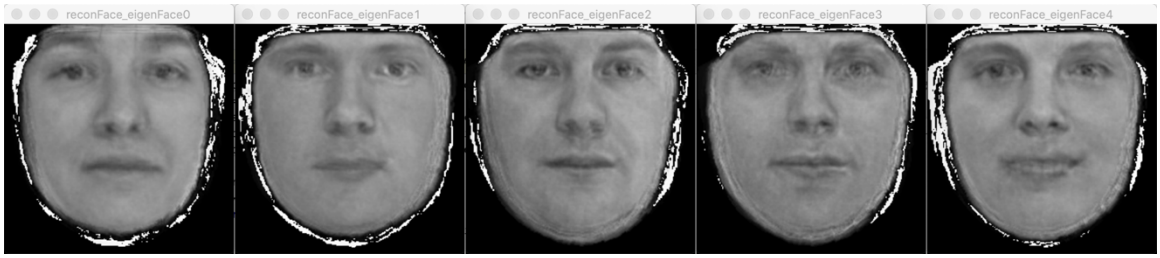
```
def reconstructTestingImages(test_dataMat, eigenVect , meanface):
    N = test_dataMat.shape[0]
    test_minus_mean = test_dataMat - meanface
    reconSystem = np.dot(test_minus_mean, eigenVect)
    reconFace = meanface + np.dot(reconSystem, np.transpose(eigenVect))

    Error = []
    row, col = reconFace.shape
    for i in range(row):
        sub = (reconFace[i,:]-test_dataMat[i,:]) ** 2
        error_sum = np.sum(sub)
        Error.append(error_sum)
    reconstructionError = np.sum(Error) / N
    print("reconstructionError: ",reconstructionError)

    return reconFace, Error, reconstructionError
```

Project the centralized test sets onto the eigen faces to get the reconstruction faces and reconstruction error.

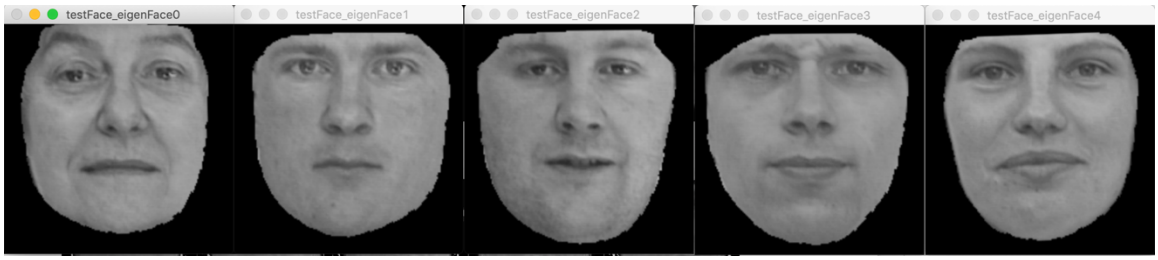
Reconstructed Faces (K=30, Reconstruction error=): 15581741):



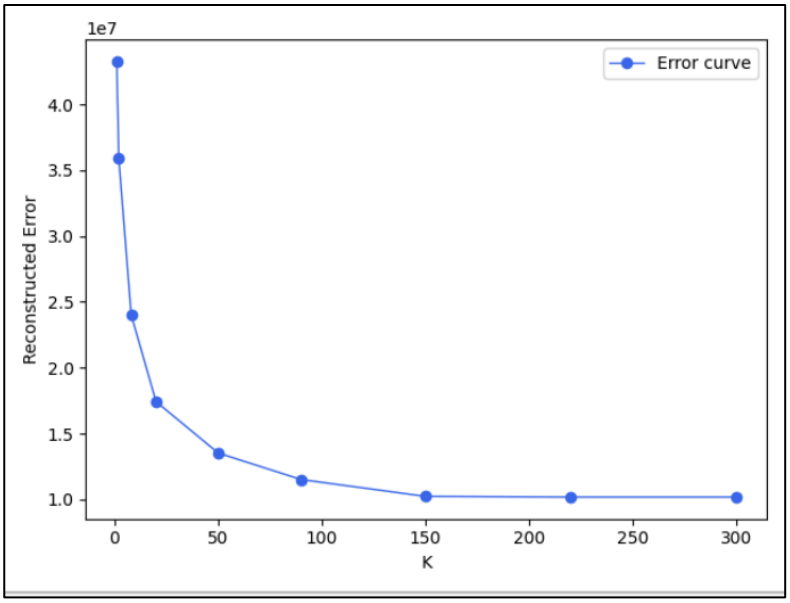
Reconstructed Faces (K=150, Reconstruction error = 10232190):



Testing Faces:



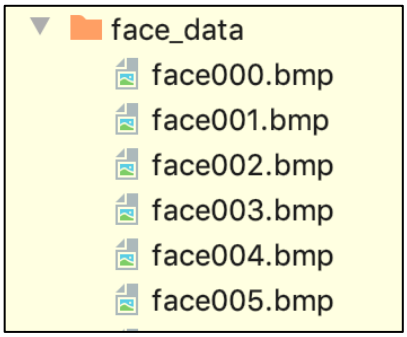
(3) Reconstruction Errors Curve:



The x-axis is K values, and the y-axis is reconstruction error.

Image reading instruction：

```
def loadImage(imgname):
    imgname = "face_data/" + imgname + ".bmp"
    pre_image = cv2.imread(imgname)
    gray_image = cv2.cvtColor(pre_image, cv2.COLOR_BGR2GRAY)

    vect = np.zeros((1, 65536))
    for i in range(256):
        for j in range(256):
            vect[0,i*256 + j] = gray_image[i,j]

    return vect
```



The original images needs to be placed in the face_data folder in the project directory.

Source code:

```
from PIL import Image
import cv2
import numpy as np
import matplotlib.pyplot as plt


def loadImage(imgname):
    imgname = "face_data/" + imgname + ".bmp"
    pre_image = cv2.imread(imgname)
    gray_image = cv2.cvtColor(pre_image, cv2.COLOR_BGR2GRAY)

    vect = np.zeros((1, 65536))
    for i in range(256):
        for j in range(256):
            vect[0,i*256 + j] = gray_image[i,j]

    return vect

def doPCA(dataMat, K):
    N, D = dataMat.shape
    mean = np.mean(dataMat, axis=0)
    minus_mean = dataMat - mean
    covMultiplyX = np.dot(minus_mean,np.transpose(minus_mean)) * (1 / N)
    eigenVal, eigenVectMultiplyX = np.linalg.eig(covMultiplyX)
    eigenVect = np.dot(np.transpose(minus_mean), eigenVectMultiplyX)
    #print(eigenVal.shape, eigenVectMultiplyX.shape, eigenVect.shape)
    eigenValRank = np.argsort(eigenVal)
    eigenValRank = eigenValRank[:-(K+1):-1]
    # print("val",eigenVal.size,"vec",eigenVect.shape)
    eigenVect_norm = eigenVect / np.linalg.norm(eigenVect, axis=0)
    eigenVectByK = eigenVect[:, eigenValRank]
    eigenVectByK_norm = eigenVect_norm[:, eigenValRank]

    return eigenVectByK, eigenVectByK_norm, mean

def visualizeEigenFace(eigenVect,times):
    row,col = eigenVect.shape
    time = 0
    eigenFaces = []
```

```
    imgnames = []
    while time < times:
        imgname = "eigenFace"+str(time)
        eigenFace = np.zeros((256, 256))
        for i in range(256):
            for j in range(256):
                eigenFace[i,j] = int(eigenVect[i*256+j, time])
        eigenFace = eigenFace.astype('uint8')
        # row, col = eigenFace.shape
        # for i in range(row):
        #     for j in range(col):
        #         print(eigenFace[i, j])
        eigenFaces.append(eigenFace)
        imgnames.append(imgname)
        time = time + 1

    return imgnames,eigenFaces


def reconstructTestingImages(test_dataMat, eigenVect , meanface):
    N = test_dataMat.shape[0]
    test_minus_mean = test_dataMat - meanface
    reconSystem = np.dot(test_minus_mean, eigenVect)
    reconFace = meanface + np.dot(reconSystem, np.transpose(eigenVect))

    Error = []
    row, col = reconFace.shape
    for i in range(row):
        sub = (reconFace[i,:]-test_dataMat[i,:]) ** 2
        error_sum = np.sum(sub)
        Error.append(error_sum)
    reconstructionError = np.sum(Error) / N
    #print("reconstructionError: ",reconstructionError)

    return reconFace, Error, reconstructionError

def drawingErrorsCurve(training_dataMat, test_dataMat):
    klist = [1,2,8,20,50,90,150,220,300]
    reconErrorList = []
    for K in range(np.size(klist)):
        k = klist[K]
```

```
        eigenface, eigenVect, meanface = doPCA(training_dataMat, k)
        reconFace, errorList, reconError = reconstructTestingImages(test_dataMat,
eigenVect, meanface)
        print(reconError)
        reconErrorList.append(reconError)
    plt.plot(klist, reconErrorList,'ro-',color='#4169E1',linewidth=1,
label='Error curve')
    plt.legend(loc="upper right")
    plt.xlabel('K')
    plt.ylabel('Reconstructed Error')
    plt.show()


if __name__ == "__main__":
    # Data preprocessing
    traning_num = 157
    test_num = 20
    # training data
    training_dataMat = np.zeros((traning_num, 65536))
    for i in range(traning_num):
        if i != 103:  # NO.103 picture is missing
            num = str(i).zfill(3)
            filename = "face"+num
            vect = loadImage(filename)
            training_dataMat[i,]=vect
    # 157 X 65536
    print(training_dataMat.shape)

    # Doing PCA and return eigenVects
    eigenface, eigenVect, meanface = doPCA(training_dataMat, 30)

    # Visualize the number of 10 eigen faces
    eigennames, eigenFaces =  visualizeEigenFace(eigenface,10)
    labelname = "eigeenFace_"
    for e in range(10):
        eigennames[e] = labelname + eigennames[e]
        cv2.imshow(eigennames[e], eigenFaces[e])

    # obtaining test data
    test_dataMat = np.zeros((test_num, 65536))
    for i in range(test_num):
```

```
        num = str(i+traning_num).zfill(3)
        filename = "face"+num
        vect = loadImage(filename)
        test_dataMat[i,]=vect
    # 20 X 65536
    print(test_dataMat.shape)

    # Reconstruct testing images
    reconFace, errorList, reconError =
reconstructTestingImages(test_dataMat,eigenVect,meanface)
    print(reconFace.shape)

    # Visualizing test faces
    testFaceNum = 5
    testnames, reconFaces = visualizeEigenFace(np.transpose(test_dataMat),
testFaceNum)
    labelname = "testFace_"
    for t in range(testFaceNum):
        testnames[t] = labelname + testnames[t]
        cv2.imshow(testnames[t], reconFaces[t])
    # Visualizing reconstucted faces
    reconFaceNum = 5
    reconnames,reconFaces =
visualizeEigenFace(np.transpose(reconFace),reconFaceNum)
    labelname = "reconFace_"
    for r in range(reconFaceNum):
        reconnames[r] = labelname + reconnames[r]
        cv2.imshow(reconnames[r], reconFaces[r])
    # Visualizing reconstucted error
    print("Reconstucted error(K=30):",reconError)

    # Visualizing mean face
    mean_eigenFace = np.zeros((256, 256))
    for i in range(256):
        for j in range(256):
            mean_eigenFace[i,j] = int(meanface[i*256+j])
    mean_eigenFace = mean_eigenFace.astype('uint8')
    # cv2.imshow("meanface", mean_eigenFace)

    # Drawing Errors Curve
    drawingErrorsCurve(training_dataMat,test_dataMat)
```

```
cv2.waitKey()
cv2.destroyAllWindows()
```