Principal Component Analysis

Code

```
import numpy as np
import os
from PIL import Image
import numpy as np
import PIL
import re
from matplotlib import pyplot as plt
from sklearn.preprocessing import normalize

def read_pgm(filename, byteorder='>'):
    with open(filename, 'rb') as f:
        buffer = f.read()
    try:
        header, width, height, maxval = re.search(
            b"(^P5\s(?:\s*#.*[\r\n])*"
            b"(\d+)\s(?:\s*#.*[\r\n])*"
            b"(\d+)\s(?:\s*#.*[\r\n])*"
            b"(\d+)\s(?:\s*#.*[\r\n]\s)*)", buffer).groups()
    except AttributeError:
        raise ValueError("Not a raw PGM file: '%s'" % filename)
    return np.frombuffer(buffer,
                        dtype='u1' if int(maxval) < 256 else byteorder+'u2',
                        count=int(width)*int(height),
                        offset=len(header)
                        ).reshape((int(height), int(width)))

features = []

for i in range(40):
    for file in os.listdir("gallery/s" + str(i+1)):
        image = read_pgm("gallery/s" + str(i+1) + '/' + file, byteorder='<')
        # print(np.shape(image))
        features.append(image.flatten())

# making data mean centered
N = len(features)
features = np.asarray(features)
features = np.reshape(features, (200,112*92))
mean = np.asarray([np.mean(features, axis=0)])
X = features - mean
eigenValThreshold = 200
```

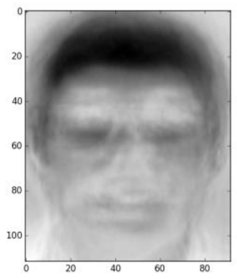
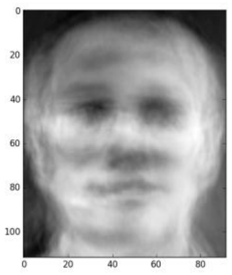
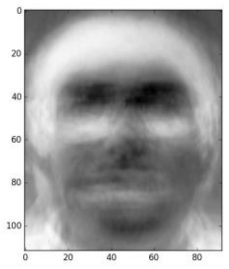
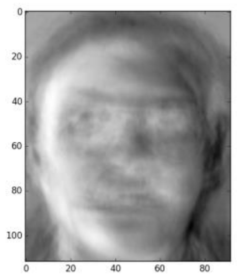
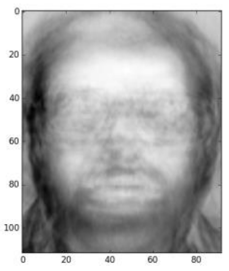
```
# EigenValue computation using low dimensional matrix
eignMat = (np.dot(X, X.T))/float(N)
eigenValues, init_eigenVec = np.linalg.eig(eignMat)
eigenValues = np.abs(eigenValues)

# calculating eigen vectors from data matrix
eigenVectors = np.dot(X.transpose(), init_eigenVec)
eigenVectors = eigenVectors.transpose()

# sorting eigen values in descending order
index = eigenValues.argsort()[::-1]
eigenValues = eigenValues[index]
eigenVectors = eigenVectors[index,:]

for count in range(len(eigenVectors)):
    eigenVectors[count] = eigenVectors[count]/float(np.linalg.norm(eigenVectors[count]))

# using only required eigen values
maxEigens = eigenValues[0:eigenValThreshold]
maxVectors = eigenVectors[0:eigenValThreshold]

# Visualizing top 5 eigen-faces
I1 = np.reshape(eigenVectors[0],  (112,92))
I2 = np.reshape(eigenVectors[1],  (112,92))
I3 = np.reshape(eigenVectors[2],  (112,92))
I4 = np.reshape(eigenVectors[3],  (112,92))
I5 = np.reshape(eigenVectors[4],  (112,92))


plt.figure(1)
plt.imshow(I1, cmap='Greys_r')
plt.show()

plt.figure(2)
plt.imshow(I2, cmap='Greys_r')
plt.show()

plt.figure(3)
plt.imshow(I3, cmap='Greys_r')
plt.show()

plt.figure(4)
plt.imshow(I4, cmap='Greys_r')
plt.show()

plt.figure(5)
plt.imshow(I5, cmap='Greys_r')
plt.show()


# Variance versus dimensions graph
totalVar = float(sum(eigenValues))
percentageVar = []
sum1 = 0
for value in eigenValues:
    sum1 = sum1 + value
    percentageVar.append((sum1*100)/totalVar)

dimensions = range(1,201)

plt.plot( dimensions, percentageVar)
plt.title('Percebtage var VS Dimensions plot')
plt.ylabel('Variance Percentage')
plt.xlabel('Dimensions considered')

plt.show()


for x in percentageVar:
    if x > 95 and x < 95.1:
        dim = percentageVar.index(x) + 1

print ("Number of dimensions covering 95% of variance = ", dim)


# Reconstructing the image ‘face input 1.pgm’ and calculating mean squared error for different
dimensions
def meanSquaredError(originalImage, reconstructedImage):
    return np.dot((originalImage - reconstructedImage), (originalImage - reconstructedImage))


testImageOne = np.reshape(read_pgm("face_input_1.pgm", byteorder='<'),(112*92) )
mean1 = mean[0]
testImageOne = testImageOne - mean1

# reconstructing using only one (max) eigen value
eigenValThreshold = 1
Eigens = eigenValues[0:eigenValThreshold]
maxVectors = eigenVectors[0:eigenValThreshold]

resultingImageOne = np.dot(np.dot(maxVectors, testImageOne.transpose()).transpose(), maxVectors)

I1 = np.reshape(resultingImageOne,  (112,92))
plt.imshow(I1, cmap='Greys_r')
plt.show()
```

```
print ("Mean squared error for the max eigen value = ", meanSquaredError(testImageOne,
resultingImageOne))

#reconstructing using top 15 eigen values
eigenValThreshold = 15
maxEigens = eigenValues[0:eigenValThreshold]
maxVectors = eigenVectors[0:eigenValThreshold]

resultingImageOne = np.dot(np.dot(maxVectors, testImageOne.transpose()).transpose(), maxVectors)

I1 = np.reshape(resultingImageOne,  (112,92))
plt.imshow(I1, cmap='Greys_r')
plt.show()
print ("Mean squared error for the top 15 eigen values = ", meanSquaredError(testImageOne,
resultingImageOne))

#reconstructing using all the eigen values
eigenValThreshold = 200
maxEigens = eigenValues[0:eigenValThreshold]
maxVectors = eigenVectors[0:eigenValThreshold]

resultingImageOne = np.dot(np.dot(maxVectors, testImageOne.transpose()).transpose(), maxVectors)
I1 = np.reshape(resultingImageOne,  (112,92))
plt.imshow(I1, cmap='Greys_r')
plt.show()
print ("Mean squared error for all eigen values = ", meanSquaredError(testImageOne,
resultingImageOne))


# Graph showing mean squared error for different dimensions for image 1
meanSquaredErrors = []
for count in range(1, 201):
   eigenValThreshold = count
   maxEigens = eigenValues[0:eigenValThreshold]
   maxVectors = eigenVectors[0:eigenValThreshold]
   resultingImageOne = np.dot(np.dot(maxVectors, testImageOne.transpose()).transpose(), maxVectors)
   meanSquaredErrors.append(meanSquaredError(testImageOne, resultingImageOne))

dimensions = range(1,201)

plt.plot( dimensions, meanSquaredErrors)
plt.title('meanSquaredErrors VS Dimensions plot')
plt.ylabel('meanSquaredErrors')
plt.xlabel('Dimensions considered')

plt.show()
```

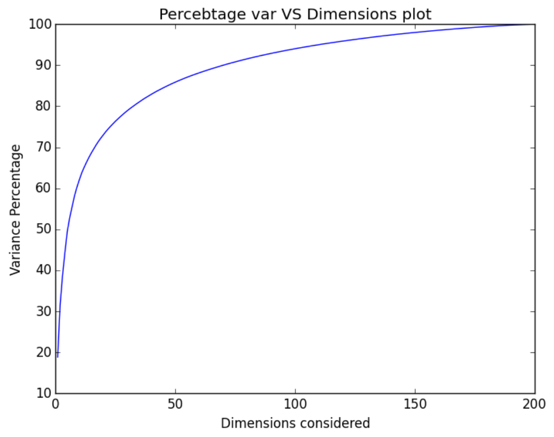
```
# Reconstructing the image ‘face input 1.pgm’ and calculating mean squared error for different
dimensions
testImageTwo = np.reshape(read_pgm("face_input_2.pgm", byteorder='<'),(112*92) )
mean2 = mean[0]
testImageTwo = testImageTwo - mean2

#reconstructing using max eigen value
eigenValThreshold = 1
maxEigens = eigenValues[0:eigenValThreshold]
maxVectors = eigenVectors[0:eigenValThreshold]

resultingImageTwo = np.dot(np.dot(maxVectors, testImageTwo.transpose()).transpose(), maxVectors)

I1 = np.reshape(resultingImageTwo,  (112,92))
plt.imshow(I1, cmap='Greys_r')
plt.show()
print ("Mean squared error for the max eigen value = ", meanSquaredError(testImageOne,
resultingImageOne))

#reconstructing using top 15 eigen values
eigenValThreshold = 15
maxEigens = eigenValues[0:eigenValThreshold]
maxVectors = eigenVectors[0:eigenValThreshold]

resultingImageTwo = np.dot(np.dot(maxVectors, testImageTwo.transpose()).transpose(), maxVectors)

I1 = np.reshape(resultingImageTwo, (112,92))
plt.imshow(I1, cmap='Greys_r')
plt.show()
print ("Mean squared error for the max 15 eigen values = ", meanSquaredError(testImageOne,
resultingImageOne))

#reconstructing using top all eigen values
eigenValThreshold = 200
maxEigens = eigenValues[0:eigenValThreshold]
maxVectors = eigenVectors[0:eigenValThreshold]

resultingImageTwo = np.dot(np.dot(maxVectors, testImageTwo.transpose()).transpose(), maxVectors)

I1 = np.reshape(resultingImageTwo,  (112,92))
plt.imshow(I1, cmap='Greys_r')
plt.show()
print ("Mean squared error for all the eigen values = ", meanSquaredError(testImageOne,
resultingImageOne))

# Graph showing mean squared error for different dimensions for image 2
meanSquaredErrors = []
for count in range(1, 201):
    eigenValThreshold = count
    maxEigens = eigenValues[0:eigenValThreshold]
    maxVectors = eigenVectors[0:eigenValThreshold]
    resultingImageTwo = np.dot(np.dot(maxVectors, testImageTwo.transpose()).transpose(), maxVectors)
    meanSquaredErrors.append(meanSquaredError(testImageTwo, resultingImageTwo))

dimensions = range(1,201)

plt.plot( dimensions, meanSquaredErrors)
plt.title('meanSquaredErrors 2 VS Dimensions plot')
plt.ylabel('meanSquaredErrors')
plt.xlabel('Dimensions considered')

plt.show()
```

Results

Top 5 eigen faces (not in order)

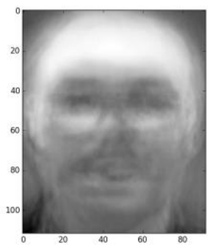






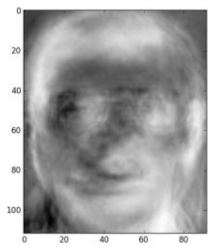
Graph for total variance versus dimensions

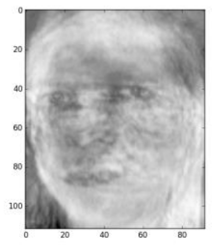


Number of dimensions required to cover 95% of variance : 110

Reconstruction for test input 1 using – top 1, top 15, all eigen values

1. 

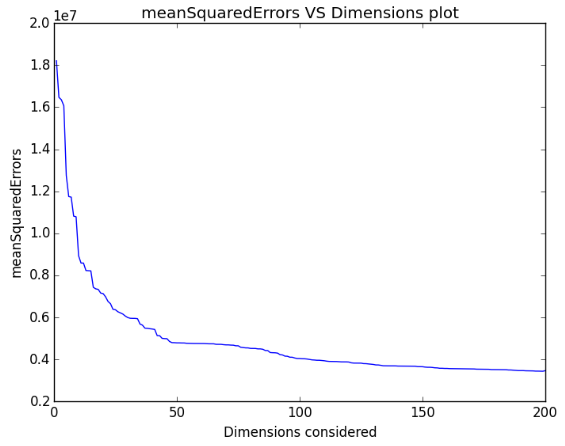
2. 

3. 

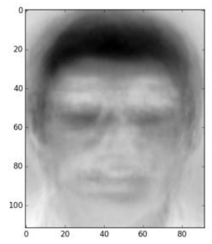
Mean Squared Error values for above reconstructions:

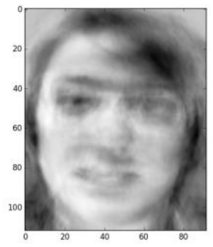
1. Top eigen value : 18209665.5288
2. Top 15 eigen values : 8212525.00992
3. All eigen values : 3465867.97964

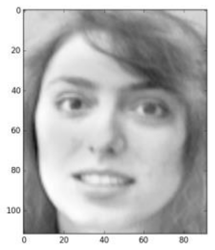
Mean squared error graph as dimension is varied (test input 1)



Reconstruction for test input 2 using – top 1, top 15, all eigen values

1. 

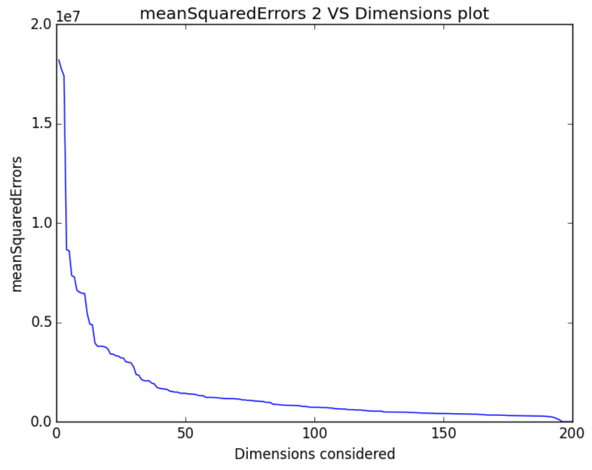
2. 

3. 

Mean Squared Error values for above reconstructions:

4. Top eigen value : 3465867.97964
5. Top 15 eigen values : 3465867.97964
6. All eigen values : 3465867.97964

Mean squared error graph as dimension is varied (test input 2)



The second test image is reconstructed with more accuracy than the first one because it is already present in the training data set and eigen vectors were calculated with that image as one of the vectors.