**Name – Sayem Lincoln**

**PID -A5420735**

**CSE 404 Homework 10 Report**

**PART1:**

**Source code part a:**

In this part, I implemented PCA algorithm these are following steps I have done:

* Loading necessary libraries
* Loading data
* PCA is sensitive to scale so in order to make not affected by scale. I transform data into standard scalar data using Standard Scalar library
* Then I created a covariance matrix by subtracting the standard deviation from mean and then taking the transform of it. After this I am taking the dot product previous calculation and the standard deviation minus mean and divided by shape of data.
* After this I am calculating the Eigen vector and Eigen values by using numpy library “linalg.eif”
* At last we are sorting the values.

import scipy.io

from sklearn.preprocessing import StandardScaler

import numpy as np

import matplotlib.pyplot as plt

mat = scipy.io.loadmat('HW10/USPS.mat.mat')

data,label=mat['A'],mat['L']

print("Orignial Data Shape = ",data.shape)

std\_1 = StandardScaler()

std\_fit=std\_1.fit(data)

std\_X=std\_fit.transform(data)

mean\_x = np.mean(std\_X, axis=0)

cov\_matrix = (std\_X - mean\_x).T

cov\_matrix=cov\_matrix.dot((std\_X - mean\_x)) / (std\_X.shape[0]-1)

eigen\_values, eigen\_vector = np.linalg.eig(cov\_matrix)

eigen\_pairs = [(np.abs(eigen\_values[i]), eigen\_vector[:,i]) for i in range(len(eigen\_values))]

eigen\_pairs.sort()

eigen\_pairs.reverse()

print('Eigenvalues in descending order:')

for val in eigen\_pairs:

print(val[0])

**Source code part b:**

In this part we are transforming the original data into less dimension using PCA. First we are getting given principal component from Eigen values which we have calculated in PCA code. Then we are using the matrix to take the dot product with original matrix. In this way we are converting the original matrix into less dimension.

d10=10

d50=50

d100=100

d200=200

temp\_d10=[]

for i in range(d10):

temp\_d10.append(eig\_pairs[i][1].reshape(256,1))

stack\_d10=tuple(temp\_d10)

temp\_d50=[]

for i in range(d50):

temp\_d50.append(eig\_pairs[i][1].reshape(256,1))

stack\_d50=tuple(temp\_d50)

temp\_d100=[]

for i in range(d100):

temp\_d100.append(eig\_pairs[i][1].reshape(256,1))

stack\_d100=tuple(temp\_d100)

temp\_d200=[]

for i in range(d200):

temp\_d200.append(eig\_pairs[i][1].reshape(256,1))

stack\_d200=tuple(temp\_d200)

matrix\_d10= np.hstack(stack\_d10)

matrix\_d50 = np.hstack(stack\_d50)

matrix\_d100 = np.hstack(stack\_d100)

matrix\_d200 = np.hstack(stack\_d200)

transformed\_d10 = data.dot(matrix\_d10)

transformed\_d50 = data.dot(matrix\_d50)

transformed\_d100 = data.dot(matrix\_d100)

transformed\_d200 = data.dot(matrix\_d200)

print("Transformed data size = ",transformed\_d10.shape)

**Source code part c:**

In this part I am calculating the reconstruction error of each given principal component. Calculation done by taking the dot product between the transform data with number of Eigen component which we had selected

reconstruction\_10=transformed\_d10.dot(matrix\_d10.T)

reconstruction\_50=transformed\_d50.dot(matrix\_d50.T)

reconstruction\_100=transformed\_d100.dot(matrix\_d100.T)

reconstruction\_200=transformed\_d200.dot(matrix\_d200.T)

def loss(org,recons):

error = (( org- recons) \*\* 2).mean()

return error

print("Reconstruction Error of D=10 is ",loss(data,reconstruction\_10))

print("Reconstruction Error of D=50 is ",loss(data,reconstruction\_50))

print("Reconstruction Error of D=100 is ",loss(data,reconstruction\_100))

print("Reconstruction Error of D=200 is ",loss(data,reconstruction\_200))

**PART 2:**

In this part, I am calculating reconstruction error for each given principal component. We can see that reconstruction becomes less as we increase the number of principal components. The reason is that more number mean preserving more variance of original data.

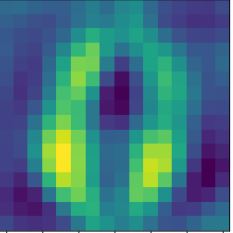
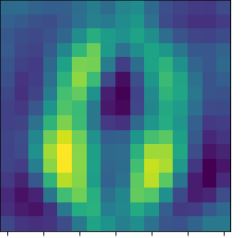
**Reconstruction Error:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Principal Components** | 10 | 50 | 100 | 200 |
|  |  |  |  |  |
| **Reconstruction Error** | 0.277 | 0.07756 | 0.02914 | 0.00409 |

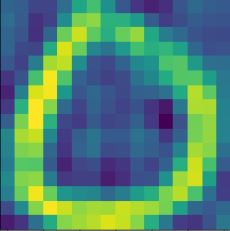
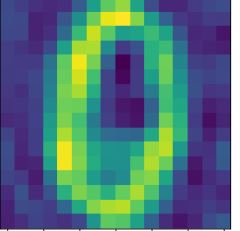
**PART 3**

In this part we are showing the reconstructed images of first 2 sample of each given principal component. The thing can be seen that if we increase the number of component then reconstructed image quality or in other words more variance of data would be preserved. That’s why 200 principal components images are looks closer to original ones.

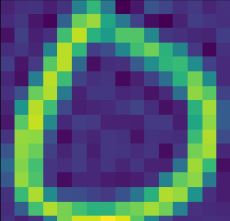
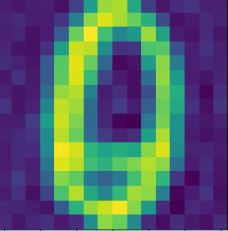
**Principal Components D=10 Reconstructed Images examples:**

****

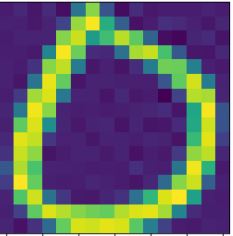
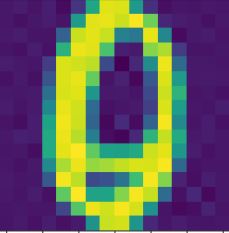
**Principal Components D=50 Reconstructed Images examples:**

****

**Principal Components D=100 Reconstructed Images examples:**

****

**Principal Components D=200 Reconstructed Images examples:**

****