```
# -*- coding: utf-8 -*-
""""
David Luong
December 24, 2019
Final Project
CPSC 483
"""
import numpy as np
import matplotlib.pyplot as plt
import scipy.io #Used to load *.mat files

def normalize(X):
    meanX = np.mean(X,axis =0)
    stdX = np.std(X,axis =0)
    meanXReplicate = np.tile(meanX,(m,1))
    stdXReplicate = np.tile(stdX,(m,1))

    X = (X - meanXReplicate) / stdXReplicate
    return X

def pca(X):
    m, n = X.shape

    # Compute covariance matrix and Eigen decomposition
    C = np.dot(X.T, X) / (m-1)
    eigen_values, eigen_vectors = np.linalg.eig(C)   # 11/ 11/11

    # Make a list of eigenvalue, eigenvector, sort and then sum
    eig_pairs = [(np.abs(eigen_values[i]), eigen_vectors[:,i])
for i in range(len(eigen_values))]
    eig_pairs.sort(key=lambda x: x[0], reverse=True)
    eig_sum = sum(eigen_values)

    #Explained Variance of top two eigen values
    var_1 = eig_pairs[0][0] / eig_sum
    var_2 = eig_pairs[1][0] / eig_sum
```

```python

    #Reduce into 2 diminsions
    matrix_w = np.hstack((eig_pairs[0][1].reshape(11,1), eig_pairs[1][1].reshape(11,1)))
    Y = X.dot(matrix_w)

    #return [reduced data, eig_1, eig_2, var_1, var_2]
    return Y, eig_pairs[0][1], eig_pairs[1][1], var_1, var_2

def svd(X):

    #Compute SVD
    U, Sigma, Vh = np.linalg.svd(X, full_matrices=False, compute_uv=True) #387/11 VAL: 11/ VECT: 11/11

    #Get Eigen Val, Vect and find largest
    eig_pairs = [(np.abs(Sigma[i]), Vh[:,i]) for i in range(len(Sigma))]
    eig_pairs.sort(key=lambda x: x[0], reverse=True)
    eig_sum = sum(Sigma)

    #Explained Variance of top two eigen values
    var_1 = eig_pairs[0][0] / eig_sum
    var_2 = eig_pairs[1][0] / eig_sum

    #Reduce into 2 diminsions
    matrix_w = np.hstack((eig_pairs[0][1].reshape(11,1), eig_pairs[1][1].reshape(11,1)))
    Y = X.dot(matrix_w)

    #return [reduced data, eig_1, eig_2, var_1, var_2]
    return Y, eig_pairs[0][1], eig_pairs[1][1], var_1, var_2

def print_data(pcaX, svdX):

    print("PCA: ")
    print("Variance: ", pcaX[3], pcaX[4])
    print("PCA principal components: \n",  pcaX[1], "\n", pcaX[2])

    print("SVD: ")
    print("Variance: ", svdX[3], svdX[4])
    print("SVD principal components: \n",  svdX[1], "\n", svdX[2])

    #check if Projected on svd and pca are matching
    if np.array_equal(pcaX[0], svdX[0]):
        print ("Projected points are matching between SVD and PCA")
    else:
        print ("Projected points are NOT matching between SVD and PCA")
    #check of principal components are matching
    if pcaX[1][0] == svdX[1][0] and pcaX[2][0] == svdX[2][0]:
        print ("Projected Components are matching between SVD and PCA")
    else: 
        print ("Projected Components are NOT matching between SVD and PCA")
    
    print("\nPCA Projected points: \n", pcaX[0], "\n")
    print("\nSVD Projected points: \n", svdX[0], "\n")

def graph(pcaX, svdX, flag):
    #True plots only SVD, False plots plots both SVD and PCA

    #Spilt data into X, Y
    x_1,y_1 = zip(*svdX[0])
    x_2,y_2 = zip(*pcaX[0])
    
    if flag:
        plt.title("Two Component SVD")
        #Set Axis Labels
        x_lab = 'Principal Component 1, match: ' +
str(round(100*svdX[3]))
        y_lab = 'Principal Component 2, match: ' +
str(round(100*svdX[4]))
        x_lab += "%"
        y_lab += "%"

        #Plot each point with labels
        for i in range(m):
            plt.scatter(x_1[i], y_1[i], s = 10,
label=data['names'][i][0])#SVD
        #Legond
        plt.legend(loc='upper left', prop={'size':1},
bbox_to_anchor=(1,1))
    else:
        plt.title("Two Component PCA and SVD")
        #Set Axis Labels
        x_lab = 'Principal Component 1'
        y_lab = 'Principal Component 2'
        plt.scatter(x_1, y_1, s = 10, label= 'SVD')
        plt.scatter(x_2, y_2, s = 10, label= 'PCA')
        plt.legend()

    plt.xlabel(x_lab)
    plt.ylabel(y_lab)
    plt.show()

""" ------------------------------------
MAIN------------------------------------ """

datafile = 'cars.mat'
data = scipy.io.loadmat( datafile )
X = data['X'] [:, 7:]
m, n = X.shape
```

```python
X = normalize(X)

#both functions returns[reduced data, eig_1, eig_2, var_1, var_2]
pcaX = pca(X)
svdX = svd(X)

print_data(pcaX, svdX)

graph(pcaX, svdX, True)
graph(pcaX, svdX, False)
```