ITU Computer and Informatics Faculty
BLG 454E Learning From Data, Spring 2018
Homework #3
Due May, 3 2018 10pm

HOMEWORK #3 SOLUTIONS

Important Note: If you see ANY mistakes please inform me via kivrakh@itu.edu.tr

1. (50 pts.) **[PCA]**

   (a) (3 pts.) What are the main motivations for reducing a dataset's dimensionality?

   - To speed up a subsequent training algorithm (in some cases it may even remove noise and redundant features, making the training algorithm perform better).
   - To visualize the data and gain insights on the most important features.
   - Simply to save space (compression).

   (b) (3 pts.) How can you evaluate the performance of a dimensionality reduction algorithm on your dataset?

   - Intuitively, a dimensionality reduction algorithm performs well if it eliminates a lot of dimensions from the dataset without losing too much information. One way to measure this is to apply the reverse transformation and measure the reconstruction error. However, not all dimensionality reduction algorithms provide a reverse transformation.
   - Alternatively, if you are using dimensionality reduction as a preprocessing step before another Machine Learning algorithm (e.g., a Random Forest classifier), then you can simply measure the performance of that second algorithm; if dimensionality reduction did not lose too much information, then the algorithm should perform just as well as when using the original dataset.

   (c) (2 pts.) What do you say about the performance of PCA in Figure **??** in terms of classification?

   - Since PCA was designed for accurate data representation, not for data classification; the directions of maximum variance is important for classification. Thus, it is clear that it mixes the class instances and does not work for this dataset.

   (d) (2 pts.) What is/are drawback(s) of PCA?

   - Some information is lost, possibly degrading the performance of subsequent training algorithms.
   - It can be computationally intensive. One problem with the preceding implementation of PCA is that it requires the whole training set to fit in memory in order the algorithm to run.
   - PCA was designed for accurate data representation, not for data classification
     - Preserves as much variance in data as possible
     - If directions of maximum variance is important for classification, will work
     - However the directions of maximum variance may be useless for classification. PCA ignores the class labels just projecting the entire set of data (without class labels) onto a different subspace.

   (e) (40 pts.) Implement a PCA projection on given the data.txt. The last attribute of the data.txt is the class label, range from 0..9.

   Recipe for Dimension Reduction with PCA

   Data $D = \{x_1, x_2, ..., x_n\}$. Each x_i is a d-dimensional vector. Wish to use PCA to reduce dimension to k. Main idea: Seek most accurate data representation in a lower dimensional space.

(a) Find the sample mean, $\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$

(b) Subtract sample mean from the data, $z_i = x_i - \mu$

(c) Compute the covariance matrix of whole dataset, $\Sigma = \sum_{i=1}^{n} z_i z_i^t$

(d) Compute eigenvectors $e_1, e_2, ..., e_k$ be the columns of matrix $E = [e_1, ..., e_k]$ corresponding to the k largest eigenvalues of Σ

(e) Let $e_1, e_2, ..., e_k$ be the columns of matrix $E = [e_1, ..., e_k]$

(f) The desired y which is the closest approximation to x is $y = E^t z$

Maximum amount of variance, as it will most likely lose less information than the other projections. Another way to justify this choice is that it is the axis that minimizes the mean squared distance between the original dataset and its projection onto that axis.

```
#Deniz Bekleyis Seven
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import random

def getDatas():
    return pd.read_csv('data.csv').values

def getBestEigenVectors(covMat):
    eigValues, eigVectors = np.linalg.eig(covMat)
    m = np.vstack((eigValues,eigVectors))
    m = m[::, m[0,].argsort()[::-1]]
    eigVectors = m[1:,0:2]
    return eigVectors

def plot():
    plt.scatter(-pca12[:,0], pca12[:,1], alpha=0.3, c=classes, s=5)
    randomSample = random.sample(range(0, len(pca12)), 200)
    for i in randomSample:
        plt.annotate(classes[i],xytext=(-pca12[i][0],pca12[i][1]),xy=(pca12[i][0],-
                                                    pca12[i][1]))
    plt.legend()
    plt.show()

data = getDatas()
classes = data[:,len(data[1,:])-1]
features = data[:,0:len(data[1,:])-1]
covMat = np.cov(np.transpose(features))
projMat = getBestEigenVectors(covMat)
pca12 = np.dot(features,projMat)
plot()
```

2. (50pts.) **[SVD]** You are going to look at compressing the given RGB image, data.jpg, through computing the singular value decomposition (SVD). Each channel (red, green, blue) has 1537×2500 pixels which is a 1537×2500 matrix A.

- (35 pts.) Find the SVD of A (one for each channel).
- (15 pts.) Display the original image and image obtained from a rank (term) of 1, 5, 20, 50 SVD approximation of A as shown in Figure **??**.
- Give all your plots in your report.

Deniz Bekleyis Seven:

My implementation for the SVD Analysis was also pretty straight forward. I am first dividing image to its rgb values, normalizing them (dividing with 255), applying the SVD function to each color matrix, ten applying the K rate with these SVD values, multiplying each color matrix with 255 to make them usual representation again, then merging them together to get reasonable image data. In my SVD function, I am first finding Transpose(A) * A, then applying eigen function in order to find s^2 and v. Then I am taking the square root of s and finding out s^2. after I get s and v, i am finding u value with using below equation.

$U = A \times V \times S^{-1}$

Then I am returning the u, s and v^T values in this function. After finding SVD, in my GetKRate function i am selecting the first K rows of each value and then applying $U \times S \times V$ in order to get new A matrix for each color matrix.

```python
#Deniz Bekleyis Seven
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt

def convertFromImage(image):
    return np.array(image)

def convertToImage(data):
    return Image.fromarray(data,'RGB')

def divideRGB(imageData):
    redData   = np.array(imageData[:, :, 0])/255
    greenData = np.array(imageData[:, :, 1])/255
    blueData  = np.array(imageData[:, :, 2])/255
    return redData, greenData, blueData

def getKRateMatrix(u, s, v, k):
    u_k = u[:,0:k]
    s_k = np.diag(s[0:k])
    v_k = v[0:k,:]
    return u_k, s_k, v_k

def mergeRGB(r,g,b):
    merged = np.zeros((len(r),len(r[1,:]),3))
    merged[:, :, 0] = r*255
    merged[:, :, 1] = g*255
    merged[:, :, 2] = b*255

    merged[merged>255] = 255
    merged[merged<0] = 0
    return np.array(merged, dtype='uint8')


def getSVD(matrix):
    a = np.array(matrix)
    aT = np.transpose(a)
    aTa = np.dot(aT,a)
    s2, v = np.linalg.eig(aTa)
    vT = np.transpose(v)
    s = np.abs(s2) ** 0.5
    sI = np.linalg.inv(np.diag(s))
    u = np.dot(a, np.dot(v, sI))
    return np.real(u), np.real(s), np.real(vT)


def getA(u,s,v):
    return np.dot(u,np.dot(s,v))

def main(imageName, k):
    img = Image.open(imageName)
    data = convertFromImage(img)
    #first we are dividing the image into its rgb matrices
    r,g,b = divideRGB(data)

    #then we apply svd to each matrix and find each of their u, s (sigma), v values
    rU, rS, rV = getSVD(r)
    gU, gS, gV = getSVD(g)
    bU, bS, bV = getSVD(b)
    #then we apply the K rate
    rUK, rSK, rVK = getKRateMatrix(rU,rS,rV,k)
    gUK, gSK, gVK = getKRateMatrix(gU,gS,gV,k)
    bUK, bSK, bVK = getKRateMatrix(bU,bS,bV,k)

    #we are getting the dot product for each u,s,v for each color matrix and combining
                                               them into image data
    #mergedData = mergeRGB(getA(rU, rS, rV), getA(gU, gS, gV), getA(bU, bS, bV))
    mergedData = mergeRGB(getA(rUK, rSK, rVK), getA(gUK, gSK, gVK), getA(bUK, bSK, bVK))

    #we are converting the image data to image object (to Pillow display)
    #newImage = convertToImage(mergedData)

    #then we return this image object
    return mergedData


def applyKRate():
    print("Calculating K=1 rate")
    img_1 = main('data.jpg',1)
    print("Calculating K=5 rate")
    img_5 = main('data.jpg',5)
    print("Calculating K=20 rate")
    img_20 = main('data.jpg',20)
    print("Calculating K=50 rate")
    img_50 = main('data.jpg',50)

    return img_1, img_5, img_20, img_50

img_1, img_5, img_20, img_50 = applyKRate()
fig, ((oi1,i1),(oi2,i5),(oi3,i20),(oi4,i50)) = plt.subplots(4,2)
img = convertFromImage(Image.open("data.jpg"))
oi1.imshow(img)
oi2.imshow(img)
oi3.imshow(img)
oi4.imshow(img)
i1.imshow(img_1)
i5.imshow(img_5)
i20.imshow(img_20)
i50.imshow(img_50)
fig.show()
```