Exercise 4 Principal Component Analysis

March 7, 2018

** Xiangyi Cheng (xxc283) **

1 Concept and Background

Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations, in our case, image data, of possibly correlated variables into a set of values of linearly uncorrelated variables called principle components. One image can be described in these principal components. One important application is to reduce the dimension of the image by reconstructing the image using first several components.

To prove the concept that reconstrction is achievable, MNIST database is used to apply PCA. The MNIST database is a large database of handwritten digits that is commonly used for training various image processing systems. The MNIST database contains 60000 training images and 10000 testing images. Training images are used for this exercise to get a higher accuracy.

2 Implement

Import the MNIST database and other useful libraries.

```
In [1]: from mnist import MNIST
        import matplotlib.pyplot as plt
        import numpy as np
        import numpy.matlib
```

Load the MNIST database inside and extract the training images. All images are labeled in a matrix called images.

```
In [2]: # load MNIST data
        mndata = MNIST('/home/liutao/python-mnist/data')
        images, labels = mndata.load_training()
```

To apply PCA in all 60000 images, it is required to reshape all the image values into one matrix. In this matrix, each row includes all the data for one image. Since the dimensions of each image are 28x28, the matrix dimensions are 60000x784.

```
In [3]: # generate a matrix and put each image information in each row
        # image size is 28*28
        datam=np.zeros((len(images),28*28))
```

```
        for index in range(0,len(images)):
                img=images[index]
                img=np.array(img,dtype='float')
                datam[index,]=img
```

The mean matrix should be computed by calculating each column mean value because it is meaningful to obtain the mean value of the same position in images.

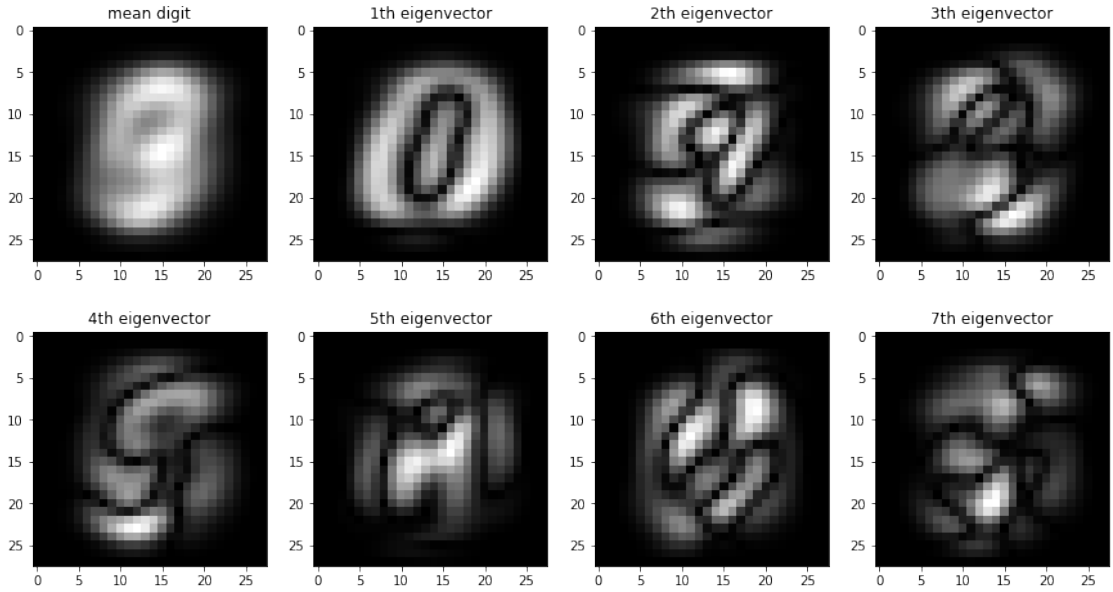
```
In [4]: # get the mean value of each column (compare images)
        mean=np.mean(datam,axis=0)
```

Mean matrix should be subtracted from the image data matrix and transpose it to let covariance of the transposed matrix fit in linalg.eig() to get the eigenvecotors and eigenvalues.

```
In [5]: # subtract mean value from the matrix and transpose it to fit in linalg.eig()
        # M is based on processed image information in each column
        M=(datam-mean).T

        # obtain the covariance matrix of M
        # latent is the eigenvalues. coeff is the normalized eigenvectors.
        # do PCA in each column
        [latent,coeff]=np.linalg.eig(np.cov(M))
        score=np.dot(coeff.T,M)
```

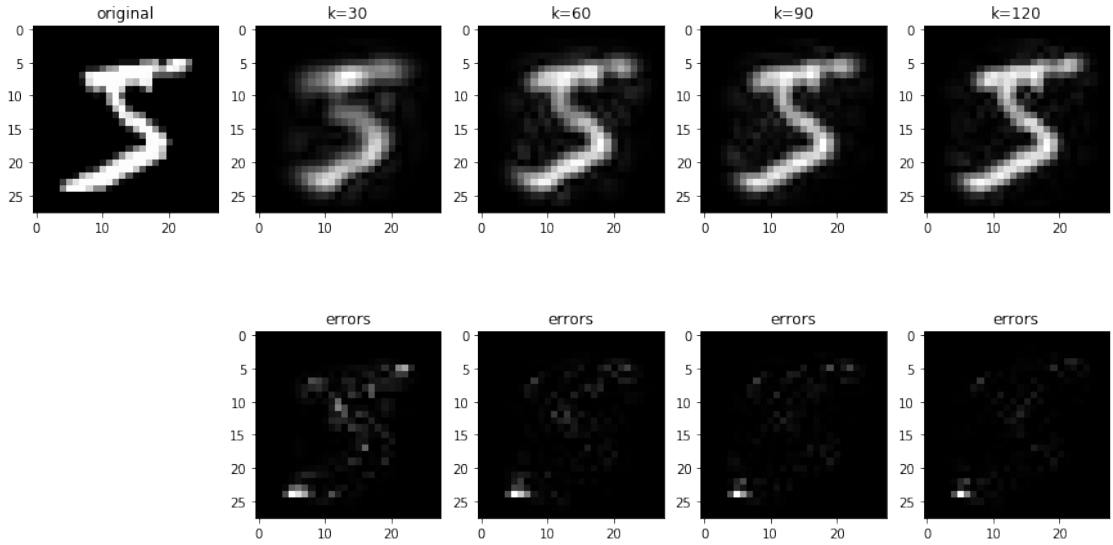
The principle components are in the coeff matrix. To plot first 7 components, reshape is needed to construct them into seven 28x28 matrices, otherwise there is no meaning within the data.

```
In [9]: plt.figure(figsize=(15,8))
        plt.subplot(2,4,1)
        plt.imshow(mean.reshape((28,28)),cmap='gray')
        plt.title('mean digit')

        components=coeff[:,0:7].reshape(28,28,7)
        for k in range(0,7):
                eigenface=abs(components[:,:,k])
                plt.subplot(2,4,k+2)
                plt.imshow(eigenface,cmap='gray')
                plt.title('%ith eigenvector'%(k+1))
        plt.show()
```



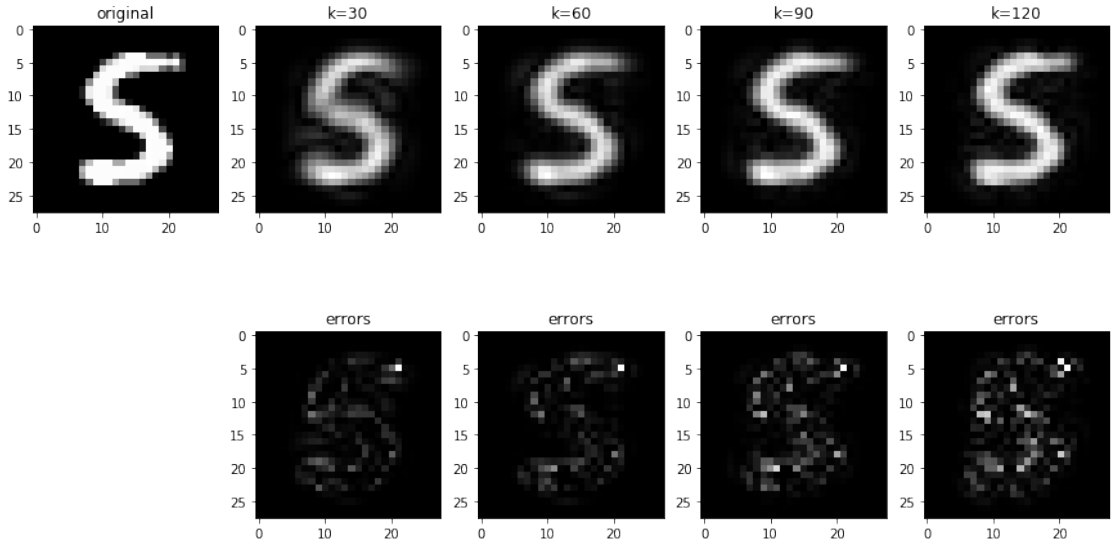
To reconstruct the digits using the sum of the first k principle components, score matrix multiply the coefficient of the components are needed. Then add the mean up to get the aproximated digit.

```
In [11]: img_id=0
         img=images[img_id]
         img=np.array(img,dtype='float')
         pixels=img.reshape((28,28))
         plt.figure(figsize=(15,8))
         plt.subplot(2,5,1)
         plt.imshow(pixels,cmap='gray')
         plt.title('original')


         for k in range(1,5):
                 # score[0:k*30,img_id] is a row vector
                 # score.shape=(28*28,60000)
                 reconstruction=np.dot(score[0:k*30,img_id],coeff[:,0:k*30].T)
                 reconstruction_tsf=abs((reconstruction+mean).reshape((28,28)))
                 diff=np.square(reconstruction_tsf-pixels)
                 plt.subplot(2,5,k+1)
                 plt.imshow(reconstruction_tsf,cmap='gray')
                 plt.title('k=%i'%(k*30))
                 plt.subplot(2,5,k+6)
                 plt.imshow(diff,cmap='gray')
                 plt.title('errors')
         plt.show()
```



Another digit is extracted to repeat the procedure.

```
In [14]: img_id=7888
         img=images[img_id]
         img=np.array(img,dtype='float')
         pixels=img.reshape((28,28))
         plt.figure(figsize=(15,8))
         plt.subplot(2,5,1)
         plt.imshow(pixels,cmap='gray')
         plt.title('original')


         for k in range(1,5):
                 # score[0:k*30,img_id] is a row vector
                 # score.shape=(28*28,60000)
                 reconstruction=np.dot(score[0:k*30,img_id],coeff[:,0:k*30].T)
                 reconstruction_tsf=abs((reconstruction+mean).reshape((28,28)))
                 diff=np.square(reconstruction_tsf-pixels)
                 plt.subplot(2,5,k+1)
                 plt.imshow(reconstruction_tsf,cmap='gray')
                 plt.title('k=%i'%(k*30))
                 plt.subplot(2,5,k+6)
                 plt.imshow(diff,cmap='gray')
                 plt.title('errors')
         plt.show()
```



3 Conclusion and Analysis

PCA is applied to get principle components of the MNIST dataset. The eigenvalues and eigenvectors are extracted to reconstruct the images by components.

The results are obvious that one image can be approximated by first several principle components. Although some information were lost during the reconstruction, we are still able to identify each processed digit even only with 30 components. Of course, the image is more clear with the components increasing due to more information. In the first example, the difference between the original image and the processed image with 120 components is tiny. So dimension reduction is totally practical to reduce the storage and computitional costing. The last but not the least, more errors showed in the second example although both are about digit 5. Therefore, PCA is a understanding approach to do image impression but how nice it is depends on the image itself.