PROJECT 4

CS691 - Machine Learning
Due Date : December 2, 2019

Hoang-Dung Bui, James Schnebly

This is the write-up for Machine Learning - Project 4. The code should run on python 3.

Chapter 1

Principle Component Analysis

This part describe how *pca.py* works.

1.1 def compute_Z(X, centering=True, scaling=False)

This function takes in a dataset, X, and returns (by default) a matrix Z which is a mutation of X where each column is centered around its mean. If scaling is set to True, each column will also be divided by it's standard deviation resulting in a standardized and centered dataset.

1.2 def compute_covariance_matrix(Z)

This function takes in Z and computes the covariance matrix which is done by multiplying Z by its transpose, Z.T.

1.3 def find_pcs(COV)

This function uses np.linalg.eig() to find eiganvalues and eiganvectors, sorts the results with argsort(), makes each eiganvector a unit vector with a helper function makeUnitVector(), and returns the sorted list of eiganvalues and eiganvectors.

1.4 def project_data(Z, PCS, L, k, var)

This function projects the top k eiganvectors and projects them onto the feature matrix Z. If k == 0, we use the cumulative variance to determine the k value. This function returns Z*, which is in a new feature space after performing PCA.

Chapter 2

Application

2.1 def load_data(input_dir)

This function loads all the images files which are in folder **Train**, then add all the image data into a matrix whose each column contains the data of an image.

The procedure of the program as following:

- read all the image files in the predefined folder with address input_dir then save them to a list *train_image_name*.
- determine the number of image and dimension of each image.
- Declare a matrix *Data* with dimensions: $len(image)*len(image[0]), len(train\_image\_name)$ to store the data of all images.
- using a for loop to flatten the image data matrix into a vector then adding to matrix *Data*

2.2 def compress_images(DATA,k)

The function project the image data into the direction with largest variance to reduce the data dimension. After completing, it will project the new data back to original size which can remove a lot of redundancy data. The procedure is following:

- transpose the input data to the right form by $Data\_T = DATA.transpose()$
- invoke function $Z = pca.compute\_Z(Data\_T, True, False)$ to standardize the data
- invoke $COV = pca.compute\_covariance\_matrix(Z)$ to calculate the covariance matrix.

- call $L, PCS = pca.find\_pcs(COV)$ function to calculate the eigenvalues and eigenvector
- call function $pr\_Data = pca.project\_data(Z, PCS, L, k, 0)$ to project data into smaller dimension.
- calculate the projected matrix
- transpose of the principles components -(eigenvector) to return the original dimension of the data.
- put the data back the original size by $X = np.matmul(pr\_Data, U\_T)$
- scaling the image data into range 0-255 $X\_inter = np.divide(X, 255/X.max())$
- check whether destination folder exists, if not open that folder
- separate the compressed data matrix back to a single image data then save to *output* folder.

This function use *Data* matrix of all image, transpose and input it to the for function in *PCA.py*.

2.3 Comparing compressed image and the original one

The original file is in Figure

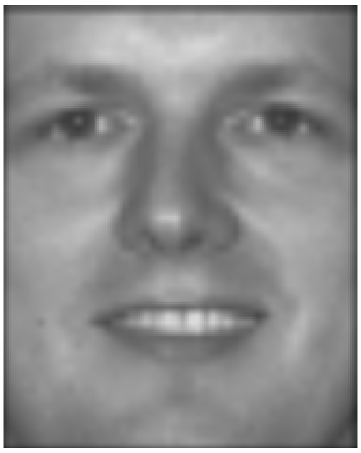

Figure 2.1: Orginal images.

k = 10
k = 100

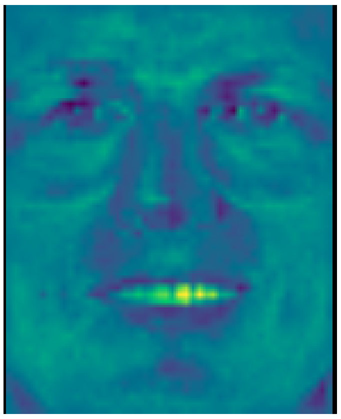

Figure 2.2: k = 100.

k = 500

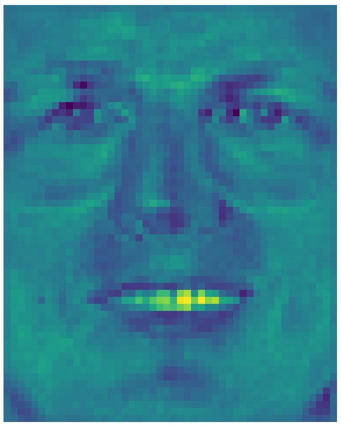

Figure 2.3: k = 500.

k = 1000

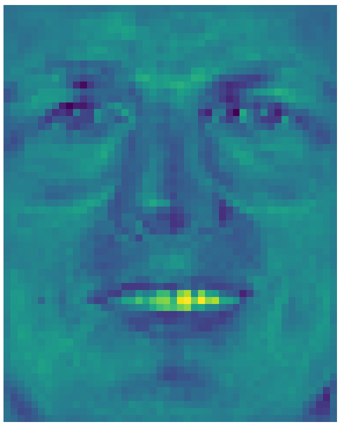

Figure 2.4: k = 1000.

k = 2000

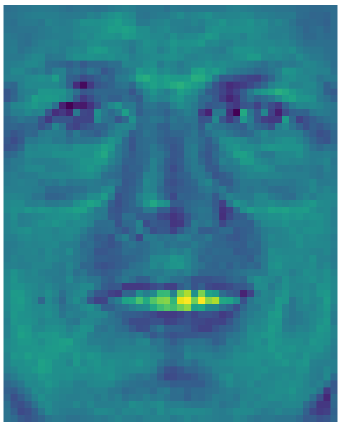

Figure 2.5: k = 2000.