Cairo University
Faculty of Engineering
Computer Engineering Department

CMPN450
Fall 2018

Pattern Recognition and Neural Networks.

Lab 6 – Principal Component Analysis

In this exercise, you will use principal component analysis (PCA) to perform dimensionality reduction. You will first experiment with an example 2D dataset to get intuition on how PCA works, and then use it on a bigger dataset of 5000 face image dataset.

The provided script, lab6.py, will help you step through this exercise.

1. Example Dataset

To help you understand how PCA works, you will first start with a 2D dataset which has one direction of large variation and one of smaller variation. The script will plot the training data. In this part of the exercise, you will visualize what happens when you use PCA to reduce the data from 2D to 1D. In practice, you might want to reduce data from 256 to 50 dimensions, say; but using lower dimensional data in this example allows us to visualize the algorithms better.

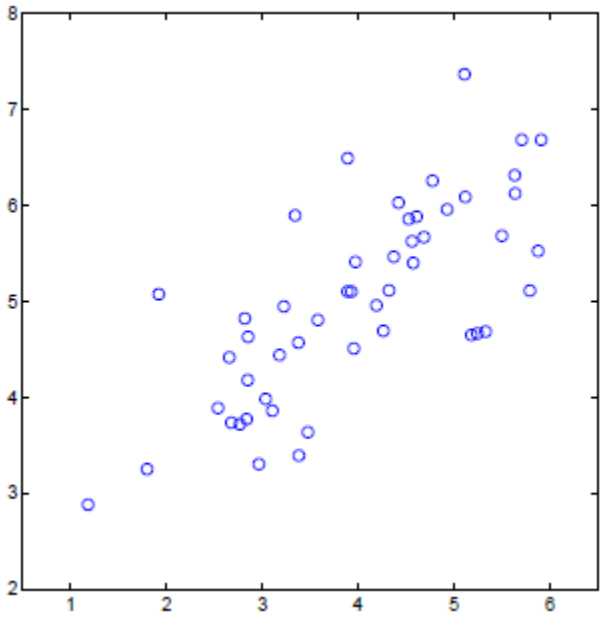

Figure 1: Example Dataset 1

2. Implementing PCA

In this part of the exercise, you will implement PCA. PCA consists of two computational steps: First, you compute the covariance matrix of the data.
Then, you use Python's SVD function to compute the eigenvectors U_1, U_2,…,U_n. These will correspond to the principal components of variation in the data.
Before using PCA, it is important to first normalize the data by subtracting the mean value of each feature from the dataset, and scaling each dimension so that they are in the same range.

After normalizing the data, you can run PCA to compute the principal components. You task is to complete the code to compute the principal components of the dataset and print these principal components.

3. Dimensionality Reduction with PCA

After computing the principal components, you can use them to reduce the feature dimension of your dataset by projecting each example onto a lower dimensional space (e.g., projecting the data from 2D to 1D). In this part of the exercise, you will use the eigenvectors returned by PCA and project the example dataset into a 1-dimensional space.

In practice, if you were using a learning algorithm such as linear regression or perhaps neural networks, you could now use the projected data instead of the original data. By using the projected data, you can train your model faster as there are less dimensions in the input.

4. Projecting the data onto the principal components

You should now complete the code in **projectData()**. Specifically, you are given a dataset X, the principal components U, and the desired number of dimensions to reduce to K. You should project each example in X onto the top K components in U.

5. Reconstructing an approximation of the data

After projecting the data onto the lower dimensional space, you can approximately recover the data by projecting them back onto the original high dimensional space. Your task is to complete **recoverData()** to project each example in Z back onto the original space and return the recovered approximation in X.

6. Visualizing the projections

After completing both **projectData()** and **recoverData(),** the script will now perform both the projection and approximate reconstruction to show how the projection affects the data. The projection effectively only retains the information in the direction given by U_1.

7. Face Image Dataset

In this part of the exercise, you will run PCA on face images to see how it can be used in practice for dimension reduction. The dataset ex7faces.mat contains a dataset X of face images, each 32x32 in grayscale. Each row of X corresponds to one face image (a row vector of length 1024). The next step in the script will load and visualize the first 100 of these face images.

Figure 2: Faces dataset

8. PCA on Faces

To run PCA on the face dataset, we first normalize the dataset by subtracting the mean of each feature from the data matrix X. The script will do this for you and then run your PCA code. After running PCA, you will obtain the principal components of the dataset. Notice that each principal component in U (each row) is a vector of length n (where for the face dataset,
n = 1024). It turns out that we can visualize these principal components by reshaping each of them into a 32x32 matrix that corresponds to the pixels in the original dataset. The script displays the first 36 principal components that describe the largest variations. If you want, you can also change the code to display more principal components to see how they capture more and more details.

9. Dimensionality Reduction

Now that you have computed the principal components for the face dataset, you can use it to reduce the dimension of the face dataset. This allows you to use your learning algorithm with a smaller input size (e.g., 100 dimensions) instead of the original 1024 dimensions. This can help speed up your learning algorithm.

The next part in the script will project the face dataset onto only the first 100 principal components. Concretely, each face image is now described by a vector of 100 dimensions.
To understand what is lost in the dimension reduction, you can recover the data using only the projected dataset. In the script, an approximate recovery of the data is performed and the original and projected face images are displayed side by side.

****This lab is adopted from Stanford's CS229 Andrew NG lecture notes.***