CSL7020 Assignment 4

Topic: Principal Component Analysis (PCA)

Wilfred Kisku (P19EE003)

17/11/2019

Problem

The goal of this assignment is to carry out dimension reduction using the concept of Principal Component Analysis *PCA*. The sub goals include operations to be carried out dimensionality reduction on particular data such as visual data or images. Also, to carry out feature reduction on a randomly generated dataset to show a visual mapping of data from 3D to 2D plane or similarly can be extended from a 2D plane to a 1D linear space.

1 Principal Component Analysis

Both Multiple Discriminant Analysis (MDA) and Principal Component Analysis (PCA) are linear transformation methods and closely related to each other. In PCA, we are interested to find the directions (components) that maximize the variance in our dataset, where in MDA, we are additionally interested to find the directions that maximize the separation (or discrimination) between different classes (for example, in pattern classification problems where our dataset consists of multiple classes. In contrast two PCA, which ignores the class labels).

In other words, via PCA, we are projecting the entire set of data (without class labels) onto a different subspace, and in MDA, we are trying to determine a suitable subspace to distinguish between patterns that belong to different classes. Or, roughly speaking in PCA we are trying to find the axes with maximum variances where the data is most spread (within a class, since PCA treats the whole data set as one class), and in MDA we are additionally maximizing the spread between classes.

Let's assume that our goal is to reduce the dimensions of a d-dimensional dataset by projecting it onto a (k)-dimensional subspace (where k<d). So, how do we know what size we should choose for k, and how do we know if we have a feature space that represents our data "well"?

Later, we will compute eigenvectors (the components) from our data set and collect them in a so-called scatter-matrix (or alternatively calculate them from the covariance matrix). Each of those eigenvectors is associated with an eigenvalue, which tell us about the "length" or "magnitude" of the eigenvectors. If we observe that all the eigenvalues are of very **similar magnitude**, this is a good indicator that our data is already in a "good" subspace. Or if some of the eigenvalues are much much higher than others, we might be interested in keeping only those eigenvectors with the much larger eigenvalues, since they contain more information about our data distribution. Vice versa, eigenvalues that are close to 0 are less informative and

we might consider in dropping those when we construct the new feature subspace.

Let us look at what it means to reduce the dimension of an image, rather that the featured dataset that we have looked earlier. As an image is defined as the collection of pixels, called a vector. One way we can think about this is in terms of a pixel basis. That is, to construct the image, we multiply each element of the vector by the pixel it describes, and then add the results together to build the image. One way we might imagine reducing the dimension of this data is to zero out all but a few of these basis vectors. For example the use of the first d basic vectors would result in a d dimension projection of the data, but that would result in throwing away $n - d$ pixel values, where n is the actual features in the images of pixels.

Apart for reduction of dimension the PCA methodology also can be used to redcue the spurious noise that might plague the image data.

PCA can be thought of as a process of choosing optimal basis functions, such that adding together just the first few of them is enough to suitably reconstruct the bulk of the elements in the dataset. The principal components, which act as the low-dimensional representation of our data, are simply the coefficients that multiply each of the elements in this series.

1.1 Algorithm

1. Take the whole dataset consisting of d-dimensional samples ignoring the class labels. In the case of the generated samples, I have created a dataset matrix of definite mean and variance that uses the inbuilt function `np.random.multivariate_normal()` function for generating the data.
2. Compute the d-dimensional mean vector (i.e., the means for every dimension of the whole dataset). Mean centering is essential for performing Principal Component Analysis, as it gives direction of variability across the mean of the samples by creating the covariance matrix
3. Compute the scatter matrix (alternatively, the covariance matrix) of the whole data set.
4. Compute eigenvectors $(e_1, e_2, e_3, ...)$ and corresponding eigenvalues $(\lambda_1, \lambda_2, \lambda_3, ...)$.
5. Sort the eigenvectors by decreasing eigenvalues and choose k eigenvectors with the largest eigenvalues to form a $d \times k$ dimensional matrix W(where every column represents an eigenvector).
6. Use this $d \times k$ eigenvector matrix to transform the samples onto the new subspace. This can be summarized by the mathematical equation: $y = W^T \times x$ (where x is a $d \times 1$-dimensional vector representing one sample, and y is the transformed $k \times 1$-dimensional sample in the new subspace.)

1.2 Result

The matrices that are evaluated are randomly generated and displayed in the python program. The important vectors generated during the execution are as listed below:

1. Mean Vector
2. Co-variance Matrix
3. Eigen values and vectors
4. Matrix W

```
Mean Vector:
[[0.54639577] [0.55725252] [0.52421495]]
Covariance Matrix:
[[1.66476027 0.25406621 0.41923147] [0.25406621 1.25197139 0.06394201] [0.41923147 0.06394201
0.88263901]]
Eigven values of covariance matrix:
[1.94379734 0.69673153 1.15884181]
Eigven vector of covariance matrix:
[[-0.86359194 -0.41475847 0.28667817] [-0.35063179 0.08545935 -0.93260605] [-0.36230693 0.
90590955 0.21922951]]
Sorting the eigenvalues in the ascending order:
114.68404298298992 68.37166656475775 41.107160388556416
Matrix W:
[[-0.86359194 0.28667817] [-0.35063179 -0.93260605] [-0.36230693 0.21922951]]
```

The figure below displays the 3D dimensional data into the 2D dimensional data points. We see here that the reduction of dimension from 3 to 2.

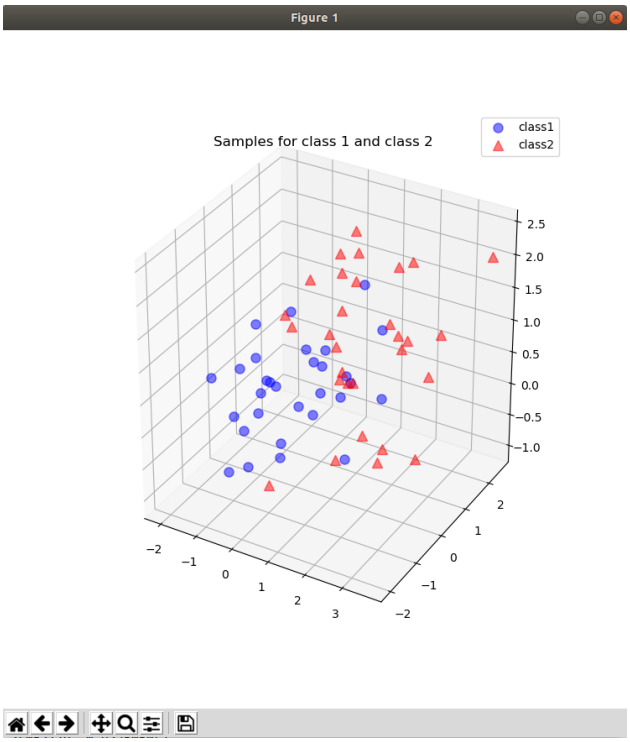


Figure 1: 3D data that needs dimensionality reduction

The following analysis is carried out and the results are obtained in regard to the reduced dimensionality of the image.

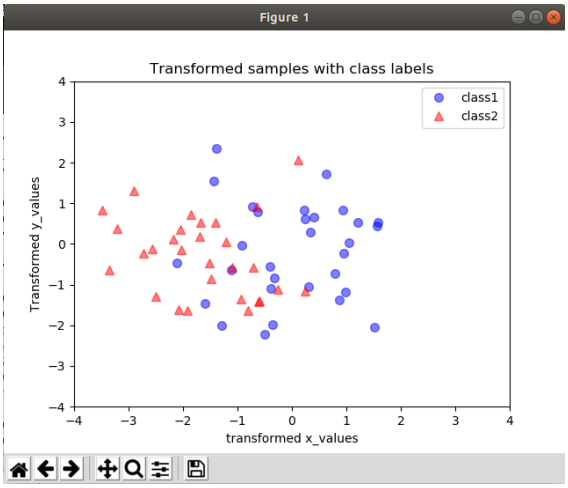


Figure 2: Transformed data points into 2D

1.3 Algorithm for Image Analysis

1. Obtain the image from the memeory using the `opencv` library for manipulation, also the `opencv` library is being used as it extracts the pixel information seamlessly in the form of `numpy` array. Extracting the information as the gray scale image with pixel intensities only.

2. Centering the image rows by finding the **mean** of the row pixels in a $n \times m$ dimensional image. Subtracting the image pixel values with the **mean** values for each of the rows.

3. Obtain the eigen value and eigen vectors of the centered and covariannce evaluated matrix using the function `np.linalg.eig()`.

4. The image eigen vectors need to be sorted with respect to the largest eigen values so as to keep most of the information present in the image as the dimension reduction takes places. Since the operation is lossy it causes a loss of information dependant on the reduction factor k.

5. The `dot` product between the selected eigen vector and the image matrix takes the image into a new dimension, this is the resultant matrix of the image that is reduced in size and a factor of $n - k$ is the loss incurred by the image. The loss in pixel information saves the memory space.

6. Finally to demonstrate that the image has lost information can be inferred by reverting back the operation and obtaining the reconstructed image by changing the transformed matrix into the same dimension of the original matrix.

7. This is obtained by adding the **mean** that had been previously subtracted for centering the image row pixel values.

8. In the figures below two images that are famous in the image processing domain, namely, `lena` and `cameraman` each of 256×256 and 512×512 respectively have been used to compress and then reconstruct. The reconstructed images visually appears to have lost information during the reconstruction.

1.4 Result

The images below show the various k factors or k components that are taken for dimensionality reduction and then conversion back to the original dimensional space.

(a) Original image

(b) k = 500

(a) k = 400

(b) k = 300

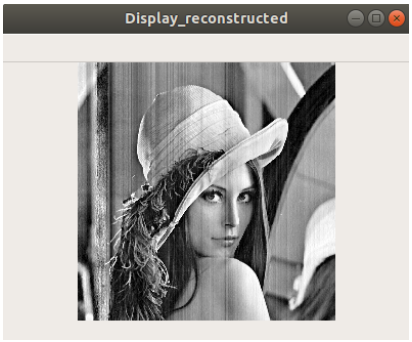
(a) k = 200

(b) k = 100

The other image that was experimented with is the `lena.png` image that is 256×256 sized image. Also, one of the weakness of PCA is that it tends to be highly affected by outliers in the data. For this reason, many robust variants of PCA have been developed, many of which act to iteratively discard data points that are poorly described by the initial components. As can be seen more predominatly in the lena image as the k values is reduced considerably.

`Scikit-Learn` contains a couple interesting variants on PCA, including `RandomizedPCA` and `SparsePCA`, both also in the `sklearn.decomposition` submodule. `RandomizedPCA`, which we saw earlier, uses a non-deterministic method to quickly approximate the first few principal components in very high-dimensional data, while `SparsePCA` introduces a regularization term that serves to enforce sparsity of the components.

(a) Original image



(b) k = 250

(a) k = 200

(b) k = 150

(a) k = 100

(b) k = 50