Principal Component Analysis

Tran Quoc De, Pham Quang Nguyen Hoang
Hanoi University of Science and Technology

May 2022

Contents

1 Some basic concepts about random vector, positive semi-definite matrix

- Given a random vector X = $\begin{bmatrix} X_1 \\ X_2 \\ ... \\ X_N \end{bmatrix}$ and its state x = $\begin{bmatrix} x_1 \\ x_2 \\ ... \\ x_N \end{bmatrix}$ then the PDF (Probability Density Function) of vector X can be formulated by :

$$f_X(x) = f_{X_1,...,X_N}(x_1, ..., x_N)$$

  with $f_{X_1,...,X_N}(x_1, ..., x_N)$ is the joint density function of $X_1, ..., X_N$

  (We can think $(X_1, ..., X_N)$ as vector X and the sample space of X is taken from $\omega_1 \times \omega_2 \times ... \times \omega_N$ with ω_n is the sample space of X_n)

- We define the expectation (or mean) vector and covariance matrix of random vector X as following:

$$\mu_X = \begin{bmatrix} E[X_1] \\ ... \\ E[X_N] \end{bmatrix}$$

$$\sum_X = E[(X-\mu_X)(X-\mu_X)^T] = \begin{bmatrix} Var(X_1) & Cov(X_1, X_2) & ... & Cov(X_1, X_N) \\ Cov(X_2, X_1) & Var(X_2) & ... & Cov(X_2, X_N) \\ ... \\ Cov(X_N, X_1) & Cov(X_N, X_2) & ... & Var(X_N) \end{bmatrix}$$

- **Definition:** A matrix $A \in \mathbb{R}^{N\times N}$ is semi-definite if $x^T Ax \geq 0$ for any $x \in \mathbb{R}^N$. A is positive if $x^T Ax > 0$ for any $x \in \mathbb{R}^N$
- **THEOREM:** Matrix A is positive semi-definite if and only if:

  $$\lambda_i(A) \geq 0$$

  for all i = 1,2,...,N and $_i(A)$ denotes the i^{th} eigenvalue of A.

  Proof :

  - If A is positive semi-definite matrix. By definition of eigenvalue, we obtain:

  $$Au_i = \lambda_i u_i$$

  where λ_i is the eigenvalue u_i is the corresponding eigenvector of A.

  Because A is positive semi-definite matrix, then $u_i^T Au_i \geq 0$ for all vector $u_i \in \mathbb{R}^n$. So we have:

  $$0 \leq u_i^T Au_i = \lambda_i ||u_i||^2$$

  and then $\lambda_i \geq 0$ for all i.

  -Conversely, if $\lambda_i \geq 0$ for all i. By eigendecomposition, we have:

  $$A = \sum_{i=1}^n \lambda_i u_i u_i^T$$

  And we can conclude that :

  $$x^T Ax = \sum_{i=1}^N \lambda_i x^T u_i u_i^T x = \sum_{i=1}^N \lambda_i (u_i^T x)^2 \geq 0$$

  for any vector $x \in \mathbb{R}^N$

  (Q.E.D)

- Now is the most crucial theorem about covariance matrix:

  THEOREM: Given the random vector X with mean μ_X and covariance $\sum_X$. Then the covariance matrix of X is a positive semi-definite matrix. Then the covariance matrix $\sum_X$ is a symmetric positive semi-definite matrix.

  Proof: Obviously, $\sum_X$ is symmetric (from the definition of $\sum_X$, and the property $Cov(X_i, X_j) = Cov(X_j, X_i)$)

In order to prove the positive semi-definiteness, we use the fact that:

$$\begin{aligned} \mathrm{v}^T \sum_X v \ &= \mathrm{v}^T E[(X-\mu_X)(X-\mu_X)^T]v \\ &= \mathrm{E}[\mathrm{v}^T (X-\mu_X)(X-\mu_X)^T v] \\ &= \mathrm{E}[\mathrm{b}^T b] = E[||b||^2] \geq 0 \end{aligned} \tag{1}$$

where b $= (X-\mu_X)^T v$

$\rightarrow$ **Corollary:** The covariance matrix $\sum_X$ are orthogonal diagonalizable and can be represented by USU^T

2 The core idea of Principal Component Analysis

Given the data set contains some vectors, our goal is representing this data set using at least coefficients as possible. For example, given the data set contains 4 vectors: $\begin{bmatrix}1\\2\end{bmatrix}, \begin{bmatrix}2\\4\end{bmatrix}, \begin{bmatrix}3\\6\end{bmatrix}, \begin{bmatrix}4\\8\end{bmatrix}$. Then we can represent each data vector by the form $\alpha \begin{bmatrix}1\\2\end{bmatrix}$ and each two-dimensional data vector can be represented by 1-dimensional coefficient α !!! The preceding process is called dimensionality reduction.

Here is more general problem: Given the dataset $\{x^{(1)}, ..., x^{(N)}\}$, we need to find the basis $\{v_i\}_{i=1}^p$ and the coefficients $\{\alpha_i\}_{i=1}^p$

- **Firstly,** we handle the case we only find the leading principal component.

  To do that, we solve the optimization problem:

  Find the argument (α, v) **to minimize** $E[||X-\alpha v||^2]$ (with $||v||_2 = 1$, we need that condition to get unique solution) (or in other words, we find (α, v) such that $x \approx \alpha v$). We solve the problem without expectation notation:

  $$(\hat{v}, \hat{\alpha}) = \arg\min_{||v||^2=1,\alpha} ||x-\alpha v||^2$$

  When the function get the minimum value, the derivative w.r.t α must be zero, so we have that:

  $$2v^T(x - \alpha \mathrm{v}) = 0 \rightarrow \alpha = v^T x \ \text{ because } (v.v^T = 1)$$

  Subtituting $\alpha = x^T.v$(because $v^T.x$ and $x^T v$ are dot product of $\vec{v}$ and $\vec{x}$ so the result is a scalar value) into the function we state before, we obtain:

  $$\begin{aligned} \arg\min_{||v||^2=1} ||x-\alpha v||^2 \ &= \arg\min_{||v||^2=1} \{x^T x - 2\alpha x^T v + \alpha v^T v\} \\ &= \arg\min_{||v||^2=1} \{-2\alpha x^T v + \alpha^2\} \\ &= \arg\min_{||v||^2=1} \{-2(x^T v) x^T v + (x^T v)^2\} \\ &= \arg\max_{||v||^2=1} \{v^T x x^T v\} \end{aligned} \tag{2}$$

Change to expectation notation we have this one:

$$\begin{aligned}\arg\min_{||v||_2=1} E||X-\alpha v||^2 &= \arg\max_{||v||_2=1} v^T E\{XX^T\}v \\ &= \arg\max_{||v||_2=1} v^T E(X-\mu)(x-\mu)^T \\ &= \arg\max_{||v||_2=1} v^T \sum_X v\end{aligned} \tag{3}$$

(with $\sum_X \in \mathbb{R}^{N\times N}$ is the covariance matrix of dataset.)

We applying **Lagrange multipliers** to handle the situation:

Let L(v,λ)=$v^T\sum_X v - \lambda(||v||^2-1)$

Taking the derivative w.r.t v and setting to zero yeilds:

$\nabla_v L(v,\lambda) = 2\sum_X v - 2\lambda v = 0$

This is equivalent to $\sum_X v = \lambda v$. So v is an eigenvector of $\sum_X$ then we substitute to (3):

$$\begin{aligned}\arg\min_{||v||_2=1} E||X-\alpha v||^2 &= \arg\max_{||v||_2=1} v^T \sum_X v \\ &= \arg\max_{||v||_2=1} v^T\lambda v \\ &= \arg\max_{||v||_2=1} \lambda \\ &= \lambda_1\end{aligned} \tag{4}$$

The equality holds when v is the eigenvector corresponding to the largest eigenvalue λ_1.

-And remember that $\alpha = v^T x$ so we can represent each 2-dimensional data vector by coefficient α (**the problem is solved for this case !!!**)

- **Now is more general problem.** The preceding process can only find the leading principal component. How can we find more ?? And from these components we can describe our data set more effectively but still keep our data simple as much as possible.

  Assume that our goal is finding p leading components (or p most crucial features of our dataset). Now our problem is a optimization problem with p constraints.

$$(\hat{\alpha_i}, \hat{v_i}) = \arg\min_{||v_1||_2=\ldots=||v_n||=1} E||X - \textstyle\sum_{i=1}^p \alpha_i v_i||^2 \tag{5}$$

  Then we can use the same idea of the previous part to solve this optimization problem : **Lagrange multiplier with p constraints**. This is out of scope of this article so we do not discuss more detailed.

  Then we can write each data vecotr as:

$$x^{(n)} = U_p.\alpha^{(n)}$$

with U_p is the matrix contains p vectors corresponding to p largest eigenvalues of the covariance matrix (or U_p is the matrix U but removing last n-p columns)

and we can find $\alpha(n)$ as:

$$\alpha^{(n)} = U_p^T x^{(n)}$$

Remark: The coordinates in vector $\alpha^{(n)}$ is the coefficients when we project vector x into p leading component vectors.

3 Application of PCA: Eigenface problem

This is one of the most important application when using PCA to analyze data.

At first, we consider N images which sizes are d (where d can be very large) pixels. Treating an image as a vector in $\mathbb{R}^d$, we obtain N of these vectors. Let us call them $x^{(1)}, ..., x^{(N)}$.

Following the process we did in the previous section, we estimate the covariance matrix by computing

$$\hat{\sum} = E[(X - \hat{\mu})(X - \hat{\mu})^T \approx \frac{1}{N} \sum_{n=1}^{N} (x^{(n)} - \hat{\mu})(x^{(n)} - \hat{\mu})^T,$$

Where the mean vector $\hat{\mu} = E[X] \approx \frac{1}{N} \sum_{n=1}^{n} x^{(n)}$. As we mentioned above, the size of the $\hat{\mu}$ is d and the size of $\hat{\sum}$ is d^2.

When we get the estimation of the covariance matrix, the next step is performing eigen-decomposition to obtain:

$$[U, S] = eig(\hat{\sum})$$

U is the matrix with the columns $\{u_i\}_{i=1}^{d}$ are the eigenvectors which obtained by the eigendecomposition. These vectors are used as the **basis** of a testing face image.

With n vectors we obtain before, we can choose the p vectors which correspond to p largest eigenvalues as a basis, and take the projection of every image of data set w.r.t to the set of leading eigenvectors we found above , this process is call **dimentionality reduction**. Specifically, for an image **x** we compute the coefficients:

$$\alpha_i = u_i^T x \qquad\qquad i = 1, ..., p$$

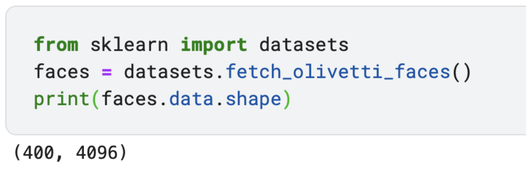
or more briefly $\alpha = U^T x$. Remarkably, the dimension of $\mathbf{x}$ is $d \times 1$, and the dimensions of α can be few as p=100. We clearly see that by PCA, a d-dimensional vector can reduce and only represent by 100-dimensional vector but still contain almost the important part. This is a huge dimensionality reduction.

The process repeats for all the samples $x^{(1)}, ..., x^{(N)}$. This gives us a collection of representation coefficients $\alpha^{(1)}, ..., \alpha^{(n)}$, where each $\alpha^{(k)}$ is 100-dimentional.

Notice that PCA decomposes a real face and written as a linear combination of these basis vector, the basis vectors u_i appear more or less ”face images” but they are the features of the faces.

These following lines of Python code can illustrate the process more intuitively:

- Fetching the datasets:

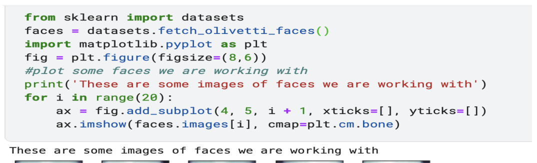


```
from sklearn import datasets
faces = datasets.fetch_olivetti_faces()
print(faces.data.shape)
```

```
(400, 4096)
```

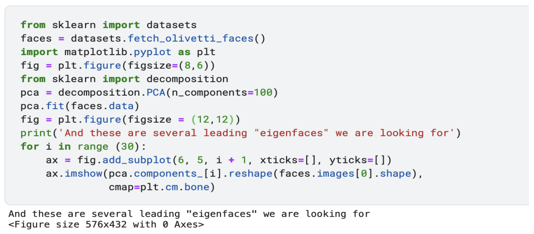
  As we can see, our data set contains a huge image set of human faces(400 pics of faces), each image has $64 \times 64 = 4096$ pixels. Each image is treated as a 4096-dimensional vector

- Handling the data :



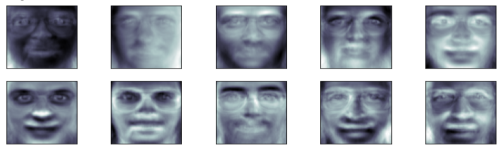
```
from sklearn import datasets
faces = datasets.fetch_olivetti_faces()
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(8,6))
#plot some faces we are working with
print('These are some images of faces we are working with')
for i in range(20):
    ax = fig.add_subplot(4, 5, i + 1, xticks=[], yticks=[])
    ax.imshow(faces.images[i], cmap=plt.cm.bone)
```

```
These are some images of faces we are working with
```



```
from sklearn import datasets
faces = datasets.fetch_olivetti_faces()
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(8,6))
from sklearn import decomposition
pca = decomposition.PCA(n_components=100)
pca.fit(faces.data)
fig = plt.figure(figsize = (12,12))
print('And these are several leading "eigenfaces" we are looking for')
for i in range (30):
    ax = fig.add_subplot(6, 5, i + 1, xticks=[], yticks=[])
    ax.imshow(pca.components_[i].reshape(faces.images[0].shape),
              cmap=plt.cm.bone)
```

```
And these are several leading "eigenfaces" we are looking for
<Figure size 576x432 with 0 Axes>
```



:

4 The downside of using PCA to compress the data

- **PCA fails when the raw data are not orthogonal.** The result is, if the data intrinsically have this othorgonality property, then PCA work very well. For example, if data points approximately form a circle with center (0,0) and radius r, then we can reform each data point by just one parameter (so the dimension shrinks !). If we use PCA in this case, it does not help very much .
- When we run our features on PCA, **the eigen comeponents are not really comprehensive**. Using PCA, our temptation is thinking each basis vector corresponding to large eigen-value is "principle component". But our method is purely mathematical operation. Using PCA on all data is independent of the class of the data set , so that lead to the inefficient of PCA when solving the classification problem.
- **PCA does not return the most "influential" components.** Sometimes, retain as much data as possible is not the optimal choice. Imagine that your purpose is researching on a disease, you have a medical data, which each data vector contains height, weight, blood pressure, etc. Using PCA can not turn out the most critical criteria.

References

[1] *Introduction to Probability for Data Science* (by Stanley H.Chan)

[2] *Linear Algebra Course* (Prof. Gilbert Strang) from MIT OpenCourseWare

[3] *Calculus Early Transcendental 7th Edition* (by James Stewart)

[4] *Linear Algebra and Learning from Data* (by Gilbert Strang)

[5] *Pattern Recognition and Machine Learning* (by Christopher M.Bishop)

[6] *Understanding Singular Value Decomposition and its Application in Data Science* (by Reza Bagheri on Medium)