
Pattern Recognition

Yukyung Choi

yk.choi@rcv.sejong.ac.kr

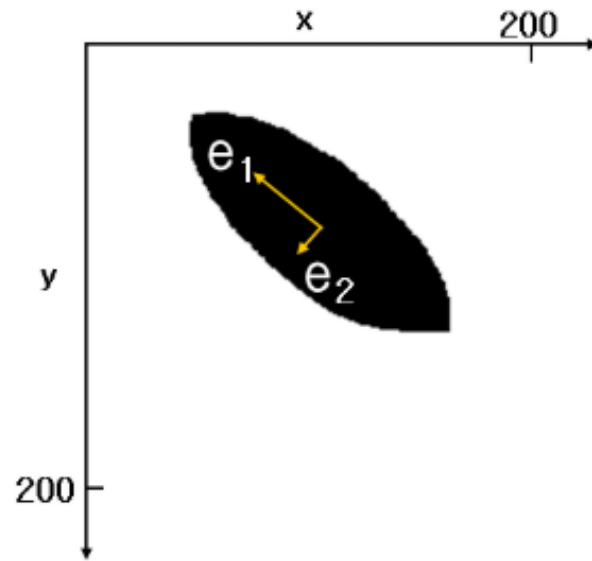
Agenda

- Principal Component Analysis (주성분 분석)
 - Using Linear Algebra
 - Statistical Analysis: Principle Components
- Use of PCA
 - Dimension Reduction
 - Data Compression
 - Noise Filtering
- Application of PCA
 - EigenFace



What is a PCA?

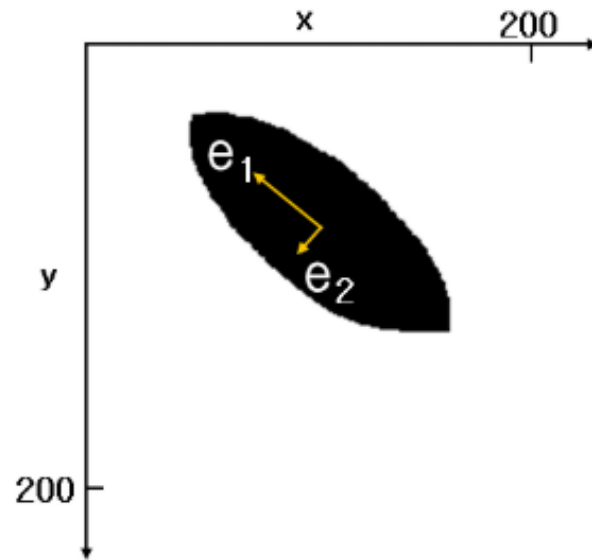
- How to find principle component in data-distribution?
 - Can you explain the following distribution with two vectors?
 - You use two principle axis such as e_1 and e_2 .





What is a PCA?

- What is the principle component?
 - The principle component is a direction vector with the largest variance, e_1 .
 - The second principle component is e_2 which is perpendicular with e_1 .



What is a PCA?

- Principal component vectors are perpendicular to each other
 - Principal component vectors can be used as a “basis”
 - Thus, data \mathbf{x} can be expressed in following equation

Principal component vectors: e_1, e_2, \dots, e_n

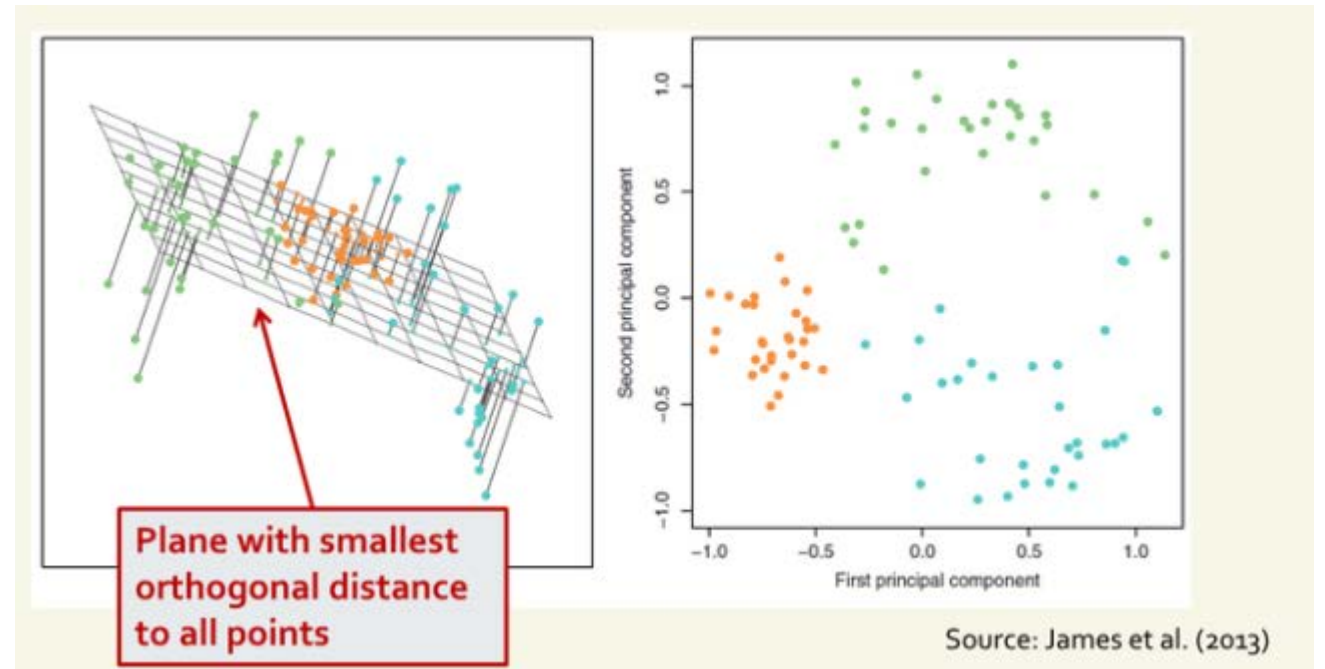
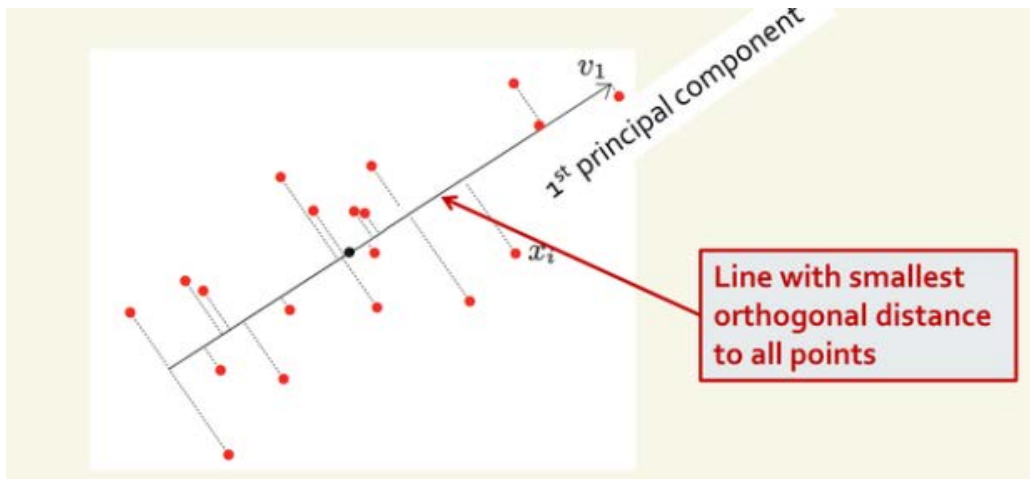
Constant coefficient: c_1, c_2, \dots, c_n

Karhunen-Loève Transform (KLT) or Hotelling transform

$$\mathbf{x} = c_1 * e_1 + c_2 * e_2 + \dots + c_n * e_n$$

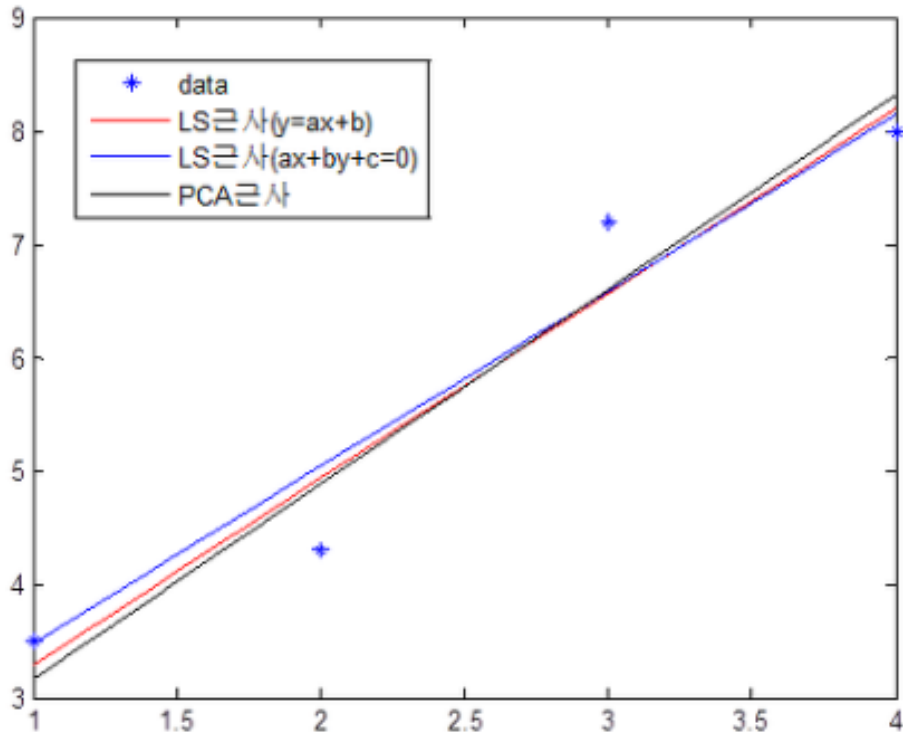
What is a PCA?

- Example of 2D, 3D Data



Use of PCA (#1)

- Line fitting with 2D data
 - The principal axis is the fitting line on 2d data



- ✓ 최소자승법은 직선과 데이터와의 거리를 최소화함
- ✓ PCA는 데이터의 분산이 가장 큰 방향을 구함

Use of PCA (#1)

- Line fitting with 3D data
 - Least Square(LS) method is not suitable
 - PCA can be used easily and efficiently

Given data are $(x_1, y_1, z_1), (x_2, y_2, z_2), \dots (x_n, y_n, z_n)$ and their average is (m_x, m_y, m_z) and principle axis is e_1 , the approximated line equation is

$$\frac{x-m_x}{a} = \frac{y-m_y}{b} = \frac{z-m_z}{c}$$

Use of PCA (#1)

- Line fitting with 3D data
 - Least Square(LS) method is not suitable
 - PCA can be used easily and efficiently

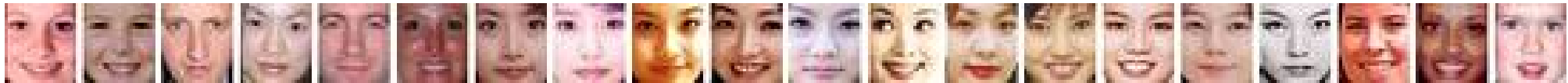
If the principal axis are e_1 and e_2 , the plane equation is

$$(e_1 \times e_2) \cdot (\mathbf{x} - \mathbf{m}) = 0$$

Where, \times is cross product, $\mathbf{x} = (x, y, z)$, $\mathbf{m} = (m_x, m_y, m_z)$.

Application: EigenFace

- Face Recognition vs EigenFace
 - Face image: $45 \times 40 = 1800$ dimensional 1d data
 - EigenFace image: 45×40 is reshaped from 1d data



Face → 45×40 pixels x 20 images



EigenFace → 45×40 pixels x 20 principle components

Application: EigenFace

- EigenFace
 - Front eigenfaces → overall shape (inter-class characteristic)
 - Middle eigenfaces → the difference information (intra-class characteristic)
 - Rear eigenfaces → noise information



$$\text{KLT} \rightarrow x = c_1 * e_1 + c_2 * e_2 + \dots c_{n-1} * e_{n-1} + c_n * e_n$$

{ {

Face information Noise information

Application: EigenFace

- Face Reconstruction



많은 수의 eigenface를 이용하면 원본 얼굴과 거의 유사한 근사(복원) 결과를 볼 수 있지만
k가 작아질수록 개인 고유의 얼굴 특성은 사라지고 공통의 얼굴 특성이 남게 된다.
k=20인 경우 원래 얼굴이 그대로 살아나지만 k=2인 경우 개인간의 구분이 거의 사라짐을 볼 수 있다.

Use of PCA (#2)

- Dimension reduction == Compression == Noise filtering

$$\text{KLT} \rightarrow x = \underbrace{c_1^* e_1 + c_2^* e_2 + \dots c_{n-1}^* e_{n-1}}_{\text{Face information}} + \underbrace{c_n^* e_n}_{\text{Noise information}} = c_1^* e_1 + c_2^* e_2 + \dots c_k^* e_k$$

Computation of PCA

- PCA란 입력 데이터들의 공분산 행렬(covariance matrix)에 대한 고유 값 분해(eigen-decomposition) 이다.
- 이 때 나오는 고유벡터(eigenvector)가 주성분 벡터로서 데이터의 분포에서 분산이 큰 방향을 나타내고, 대응되는 고유값(eigenvalue)이 그 분산의 크기를 나타낸다.

Computation of PCA

PCA

- C : covariance matrix of x
- $C = P\Sigma P^T$ (P : orthogonal, Σ : diagonal)

$$C = \begin{pmatrix} | & & | \\ e_1 & \dots & e_n \\ | & & | \end{pmatrix} \begin{pmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_n \end{pmatrix} \begin{pmatrix} \boxed{e_1^T} \\ \vdots \\ \boxed{e_n^T} \end{pmatrix}$$

← Spectral decomposition (AKA eigen-decomposition)
Singular Value Decomposition (SVD)

- P : $n \times n$ orthogonal matrix
- Σ : $n \times n$ diagonal matrix
- $Ce_i = \lambda_i e_i$
 - e_i : eigenvector of C , direction of variance
 - λ_i : eigenvalue, e_i 방향으로의 분산
 - $\lambda_1 \geq \dots \geq \lambda_n \geq 0$
- e_1 : 가장 분산이 큰 방향
- e_2 : e_1 에 수직이면서 다음으로 가장 분산이 큰 방향
- e_k : e_1, \dots, e_{k-1} 에 모두 수직이면서 가장 분산이 큰 방향

Computation of PCA

- Covariance Matrix (공분산행렬), Covariance (공분산)

$$C = \begin{pmatrix} \text{cov}(x,x) & \text{cov}(x,y) \\ \text{cov}(x,y) & \text{cov}(y,y) \end{pmatrix}$$
$$= \begin{pmatrix} \frac{1}{n} \sum (x_i - m_x)^2 & \frac{1}{n} \sum (x_i - m_x)(y_i - m_y) \\ \frac{1}{n} \sum (x_i - m_x)(y_i - m_y) & \frac{1}{n} \sum (y_i - m_y)^2 \end{pmatrix}$$

x의 분산은 x들이 평균을 중심으로 얼마나 흩어져 있는지를 나타내고, x와 y의 공분산은 x, y의 흩어진 정도가 얼마나 서로 상관관계를 가지고 흩어졌는지를 나타냄

$$\begin{aligned} \text{cov}(x,y) &= E[(x - m_x)(y - m_y)] \\ &= E[xy] - m_x m_y \end{aligned}$$

m_x 는 x의 평균, m_y 는 y의 평균, $E[]$ 는 기대값(평균)

PCA implementation from scratch python

```
1  from numpy import array
2  from numpy import mean
3  from numpy import cov
4  from numpy.linalg import eig
5  # define a matrix
6  A = array([[1, 2], [3, 4], [5, 6]])
7  print(A)
8  # calculate the mean of each column
9  M = mean(A.T, axis=1)
10 print(M)
11 # center columns by subtracting column means
12 C = A - M
13 print(C)
14 # calculate covariance matrix of centered matrix
15 V = cov(C.T)
16 print(V)
17 # eigendecomposition of covariance matrix
18 values, vectors = eig(V)
19 print(vectors)
20 print(values)
21 # project data
22 P = vectors.T.dot(C.T)
23 print(P.T)
24
```

$$C = \begin{pmatrix} \text{cov}(x,x) & \text{cov}(x,y) \\ \text{cov}(x,y) & \text{cov}(y,y) \end{pmatrix}$$
$$= \begin{pmatrix} \frac{1}{n} \sum (x_i - m_x)^2 & \frac{1}{n} \sum (x_i - m_x)(y_i - m_y) \\ \frac{1}{n} \sum (x_i - m_x)(y_i - m_y) & \frac{1}{n} \sum (y_i - m_y)^2 \end{pmatrix}$$

$$\begin{aligned} \text{cov}(x,y) &= E[(x - m_x)(y - m_y)] \\ &= E[xy] - m_x m_y \end{aligned}$$

PCA implementation from scratch python

```
1  from numpy import array
2  from numpy import mean
3  from numpy import cov
4  from numpy.linalg import eig
5  # define a matrix
6  A = array([[1, 2], [3, 4], [5, 6]])
7  print(A)
8  # calculate the mean of each column
9  M = mean(A.T, axis=1)
10 print(M)
11 # center columns by subtracting column means
12 C = A - M
13 print(C)
14 # calculate covariance matrix of centered matrix
15 V = cov(C.T)
16 print(V)
17 # eigendecomposition of covariance matrix
18 values, vectors = eig(V)
19 print(vectors)
20 print(values)
21 # project data
22 P = vectors.T.dot(C.T)
23 print(P.T)
24
```

```
1  # Principal Component Analysis
2  from numpy import array
3  from sklearn.decomposition import PCA
4  # define a matrix
5  A = array([[1, 2], [3, 4], [5, 6]])
6  print(A)
7  # create the PCA instance
8  pca = PCA(2)
9  # fit on data
10 pca.fit(A)
11 # access values and vectors
12 print(pca.components_)
13 print(pca.explained_variance_)
14 # transform data
15 B = pca.transform(A)
16 print(B)
```

Hands on Labs

- PCA Introduction

- https://colab.research.google.com/drive/11Rqpy_Lyh_9r_GYz2iPKBdA04YTTOXIZ

- PCA EigenFace

- <https://colab.research.google.com/drive/1crVmQxc4k2TT61xmNsBxpMxljCo4m2q4>

Reference

- <https://darkpgmr.tistory.com/110>
- <https://machinelearningmastery.com/calculate-principal-component-analysis-scratch-python/>
- <https://m.blog.naver.com/PostView.nhn?blogId=sw4r&logNo=221031465518&proxyReferer=https%3A%2F%2Fwww.google.com%2F>