Pattern Recognition

Yukyung Choi

yk.choi@rcv.sejong.ac.kr

Agenda

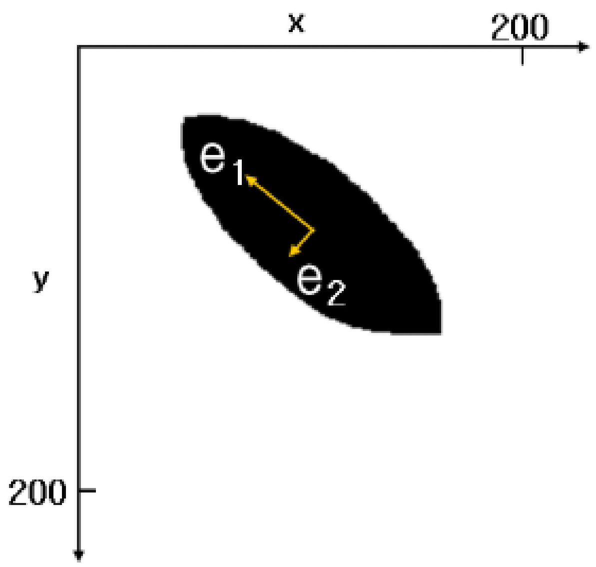
- Principal Component Analysis (주성분 분석)
  - Using Linear Algebra
  - Statistical Analysis: Principle Components

- Use of PCA
  - Dimension Reduction
  - Data Compression
  - Noise Filtering

- Application of PCA
  - EigenFace

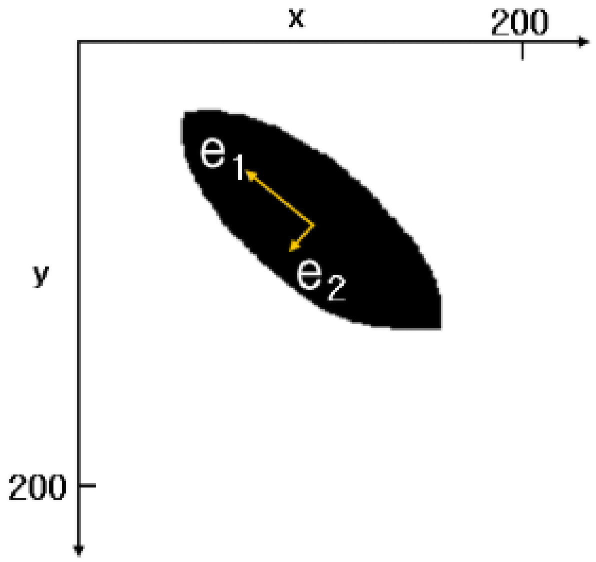
What is a PCA?

- How to find principle component in data-distribution?
  - Can you explain the following distribution with two vectors?
    - You use two principle axis such as e1 and e2.



What is a PCA?

- What is the principle component?
  - The principle component is a direction vector with the largest variance, e1.
  - The second principle component is e2 which is perpendicular with e1.



What is a PCA?

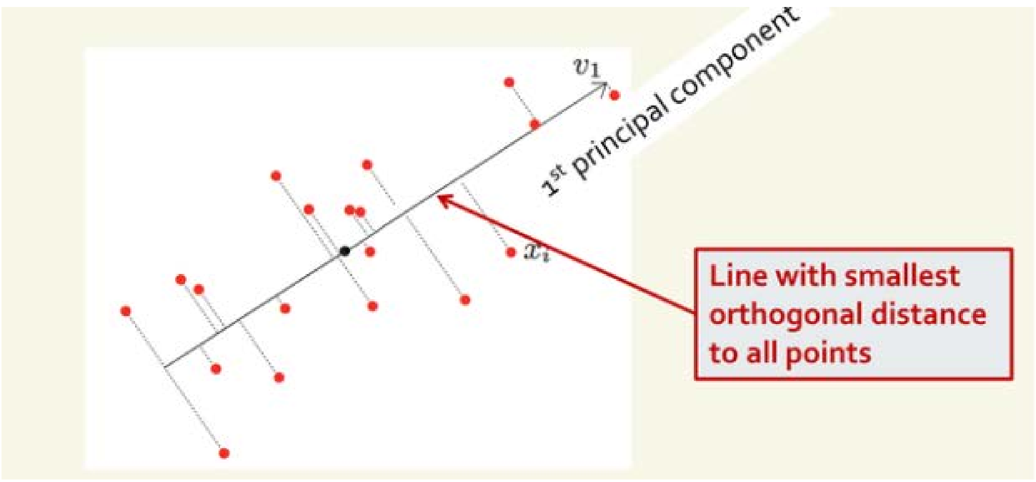
- Principal component vectors are perpendicular to each other
  - Principal component vectors can be used as a “basis”
  - Thus, data **x** can be expressed in following equation

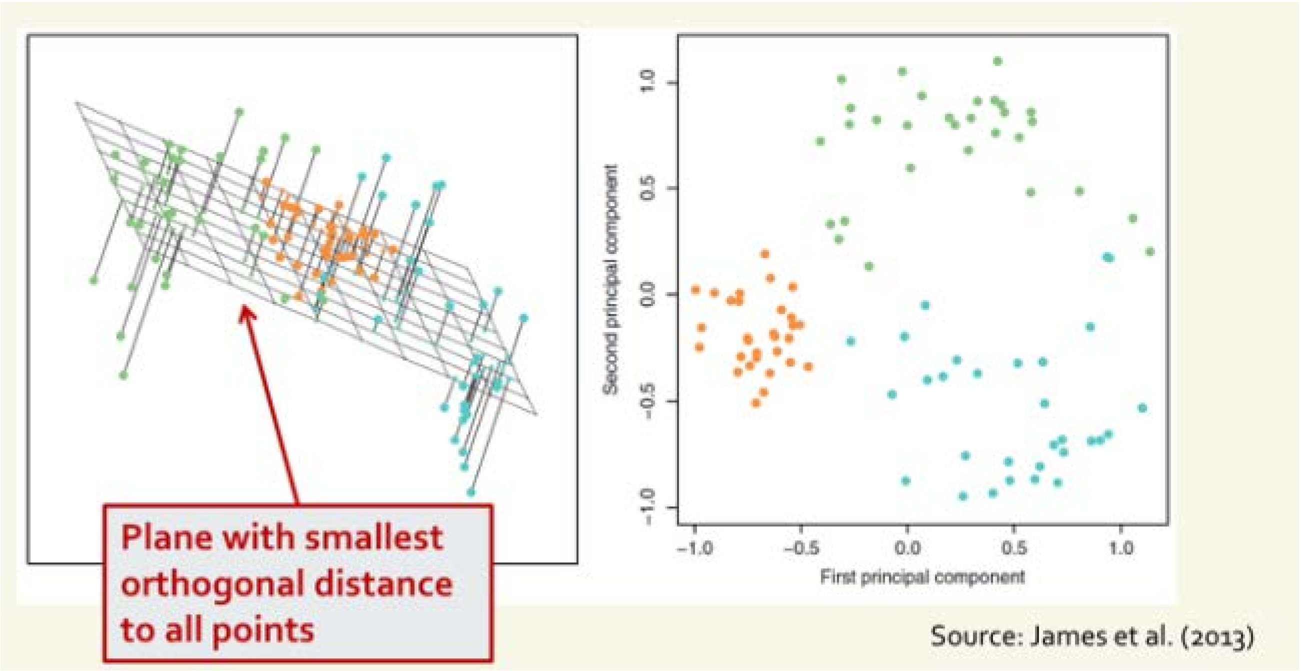
Principal component vectors: e1, e2, ..., en
Constant coefficient: c1, c2, ..., cn

Karhunen-Loève Transform (KLT) or Hotelling transform
x = c1*e1+ c2*e2+...+ cn*en

What is a PCA?

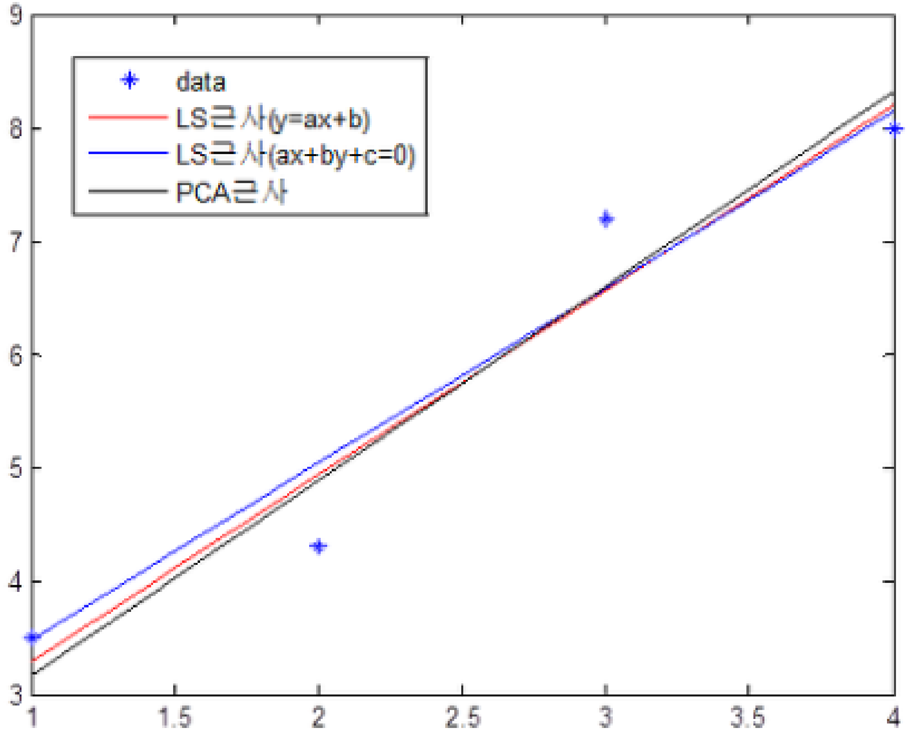
- Example of 2D, 3D Data





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 (x1,y1,z1), (x2,y2,z2), … (xn,yn,zn) and their average is (mx,my,mz) and principle axis is e1, 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

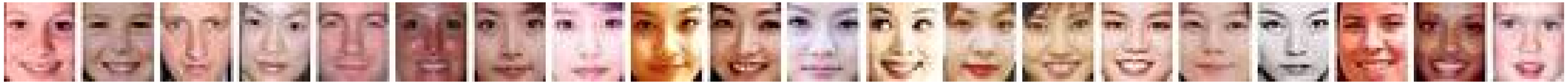
If the principal axis are e1 and e2, the plane equation is

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

Where, x is cross product, x = (x,y,z), m = (mx,my,mz).

Application: EigenFace

- Face Recognition vs EigenFace
  - Face image: 45x40=1800 dimensional 1d data
  - EigenFace image: 45x40 is reshaped from 1d data



Face ➔ 45x40 pixels x 20 images

EigenFace ➔ 45x40 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

$$\text{KLT} \Rightarrow x = \underbrace{c_1 * e_1 + c_2 * e_2}_{\text{Face information}} + \ldots \underbrace{c_{n-1} * e_{n-1} + c_n * e_n}_{\text{Noise information}}$$

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

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

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

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

- P : n×n orthogonal matrix
- Σ : n×n diagonal matrix
- $Ce_i = \lambda_i e_i$
  - e_i : eigenvector of C, direction of variance
  - λ_i : eigenvalue, e_i 방향으로의 분산
  - $\lambda_1 \geq \ldots \geq \lambda_n \geq 0$
- e_1: 가장 분산이 큰 방향
- e_2: e_1에 수직이면서 다음으로 가장 분산이 큰 방향
- e_k: $e_1, \ldots, e_{k-1}$에 모두 수직이면서 가장 분산이 큰 방향

Computation of PCA

- Covariance Matrix (공분산행렬), Covariance (공분산)

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

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

$$cov(x,y)=E[(x-m_x)(y-m_y)]$$
$$=E[xy]-m_x m_y$$

mx는 x의 평균, my는 y의 평균, E[]는 기대값(평균)

PCA implementation from scratch python

```python
from numpy import array
from numpy import mean
from numpy import cov
from numpy.linalg import eig
# define a matrix
A = array([[1, 2], [3, 4], [5, 6]])
print(A)
# calculate the mean of each column
M = mean(A.T, axis=1)
print(M)
# center columns by subtracting column means
C = A - M
print(C)
# calculate covariance matrix of centered matrix
V = cov(C.T)
print(V)
# eigendecomposition of covariance matrix
values, vectors = eig(V)
print(vectors)
print(values)
# project data
P = vectors.T.dot(C.T)
print(P.T)
```

$$C=\begin{pmatrix} cov(x,x) & cov(x,y) \\ cov(x,y) & cov(y,y) \end{pmatrix}$$

$$=\begin{pmatrix} \frac{1}{n}\sum(x_i-m_x)^2 & \frac{1}{n}\sum(x_i-m_x)(y_i-m_y) \\ \frac{1}{n}\sum(x_i-m_x)(y_i-m_y) & \frac{1}{n}\sum(y_i-m_y)^2 \end{pmatrix}$$

$$cov(x,y)=E[(x-m_x)(y-m_y)]$$
$$=E[xy]-m_x m_y$$

PCA implementation from scratch python

```python
from numpy import array
from numpy import mean
from numpy import cov
from numpy.linalg import eig
# define a matrix
A = array([[1, 2], [3, 4], [5, 6]])
print(A)
# calculate the mean of each column
M = mean(A.T, axis=1)
print(M)
# center columns by subtracting column means
C = A - M
print(C)
# calculate covariance matrix of centered matrix
V = cov(C.T)
print(V)
# eigendecomposition of covariance matrix
values, vectors = eig(V)
print(vectors)
print(values)
# project data
P = vectors.T.dot(C.T)
print(P.T)
```

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

Hands on Labs

- PCA Introduction (Lv0)
  - https://colab.research.google.com/drive/11Rqpy_Lyh_9r_GYz2iPKBdA04YTTOXlZ

- PCA implementation (Lv1)

- PCA EigenFace (Lv1)
  - https://colab.research.google.com/drive/1crVmQxc4k2TT61xmNsBxpMxljCo4m2q4

- Glass Classification with PCA – "HW" (Lv2)
  - [Problem]
    - https://www.kaggle.com/uciml/glass
  - [Solution]
    - https://www.kaggle.com/slamnz/glass-dataset-principal-components-analysis