

연세대학교 통계 데이터 사이언스 학회 ESC 23-2 SUMMER WEEK2

# Singular Value Decomposition and Its Application

[ESC 방학세션 2조] 임승현 장덕재 정석훈 김근영



# 1. Introduction

# Introduction

## Why SVD?

- 대용량 디지털 자료를 압축하여 효과적으로 정보를 저장 및 전송을 가능하게 해준다.



- 선형계의 문제를 풀기 위해 사용할 수 있는 계산 알고리즘의 기초가 된다.

Ex) ill - conditioned 혹은 nearly rank-deficient로 인해 unstable한 solution

→ SVD를 통해 stable(혹은 less sensitive)한 solution으로 만들어준다.



Figure 1



## 2. Singular Value Decomposition

# Singular Value Decomposition

## Full SVD

Rank가  $k$ 인  $m \times n$  행렬  $A$ 에 대해

$$A = U\Sigma V^T = [\mathbf{u}_1 \ \mathbf{u}_2 \ \cdots \ \mathbf{u}_k \mid \mathbf{u}_{k+1} \ \cdots \ \mathbf{u}_m] \left[ \begin{array}{cccc|cc} \sigma_1 & 0 & \cdots & 0 & 0_{k \times (n-k)} & \mathbf{v}_1^T \\ 0 & \sigma_2 & \cdots & 0 & & \mathbf{v}_2^T \\ \vdots & \vdots & \ddots & \vdots & & \vdots \\ 0 & 0 & \cdots & \sigma_k & & \mathbf{v}_k^T \\ \hline & & & & 0_{(m-k) \times k} & \mathbf{v}_{k+1}^T \\ & & & & & \vdots \\ & & & & & \mathbf{v}_n^T \end{array} \right]$$

Figure 2

$$\begin{bmatrix} \times & \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times & \times \end{bmatrix}$$

$$\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix}$$

Figure 3

- $U \in \mathbb{C}^{m \times m}$ ,  $V^T \in \mathbb{C}^{n \times n}$  이고 각각은 unitary matrices
- $\Sigma \in \mathbb{C}^{m \times n}$  는 singular values들을 주대각성분으로 가지고 나머지 원소는 0
- $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_k \geq 0$
- $\{\mathbf{u}_1, \dots, \mathbf{u}_k\}$  : left singular vectors
- $\{\mathbf{v}_1, \dots, \mathbf{v}_k\}$  : right singular vectors
- $A\mathbf{v}_j = \sigma_j \mathbf{u}_j$  가 성립

주대각성분 (Main diagonal)

# Singular Value Decomposition

Example

$$A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}$$

i)  $A^T A$ 의 eigenvalue, eigenvector 구하기

$$A^T A = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

$$\lambda^2 - 4\lambda + 3 = (\lambda - 3)(\lambda - 1)$$

$$\sigma_1 = \sqrt{\lambda_1} = \sqrt{3}, \quad \sigma_2 = \sqrt{\lambda_2} = 1$$

$$\mathbf{v}_1 = \begin{bmatrix} \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \end{bmatrix} \quad \text{and} \quad \mathbf{v}_2 = \begin{bmatrix} \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} \end{bmatrix}$$

내림차순으로 eigen value를 정렬하고 대응되는 eigen vector를 normalizing!

ii)  $\sigma, \mathbf{v}$ 를 통해  $\mathbf{u}$  구하기

$A\mathbf{v}_j = \sigma_j \mathbf{u}_j$  관계로 인해

$$\mathbf{u}_1 = \frac{1}{\sigma_1} A\mathbf{v}_1 = \frac{\sqrt{3}}{3} \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{6}}{3} \\ \frac{\sqrt{6}}{6} \\ \frac{\sqrt{6}}{6} \end{bmatrix}$$

$$\mathbf{u}_2 = \frac{1}{\sigma_2} A\mathbf{v}_2 = (1) \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} \end{bmatrix} = \begin{bmatrix} 0 \\ -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \end{bmatrix}$$



# Singular Value Decomposition

## Example

iii)  $\mathbf{u}_3$  확장하기

$U$ 는  $\mathbb{C}^{3 \times 3}$ 인 unitary 행렬인데 1개의 basis 부족!  $\rightarrow \mathbf{u}_1, \mathbf{u}_2$ 에 대해 orthonormal 한 vector 추가해주기

$$\begin{bmatrix} 2 & 1 & 1 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \mathbf{u}_3 = \begin{bmatrix} -\frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{3}} \end{bmatrix}$$

최종적으로,  $A = U\Sigma V^T$  는 아래의 형태이다. 여기서,  $U \in \mathbb{C}^{3 \times 3}$   $\Sigma \in \mathbb{C}^{3 \times 2}$   $V^T \in \mathbb{C}^{2 \times 2}$

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{6}}{3} & 0 & -\frac{1}{\sqrt{3}} \\ \frac{\sqrt{6}}{6} & -\frac{\sqrt{2}}{2} & \frac{1}{\sqrt{3}} \\ \frac{\sqrt{6}}{6} & \frac{\sqrt{2}}{2} & \frac{1}{\sqrt{3}} \end{bmatrix} \begin{bmatrix} \sqrt{3} & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{bmatrix}$$

# Singular Value Decomposition

## Full SVD

$$A = U\Sigma V^T = [\mathbf{u}_1 \ \mathbf{u}_2 \ \cdots \ \mathbf{u}_k \mid \mathbf{u}_{k+1} \ \cdots \ \mathbf{u}_m]$$

$$\left[ \begin{array}{cccc|c} \sigma_1 & 0 & \cdots & 0 & 0_{k \times (n-k)} \\ 0 & \sigma_2 & \cdots & 0 & 0_{(m-k) \times k} \\ \vdots & \vdots & \ddots & \vdots & 0_{(m-k) \times k} \\ 0 & 0 & \cdots & \sigma_k & 0_{(m-k) \times (n-k)} \end{array} \right] \left[ \begin{array}{c} \mathbf{v}_1^T \\ \mathbf{v}_2^T \\ \vdots \\ \mathbf{v}_k^T \\ \hline \mathbf{v}_{k+1}^T \\ \vdots \\ \mathbf{v}_n^T \end{array} \right]$$

Figure 2

- 행렬 A는 k개의 singular value를 가짐 (rank가 k 이므로)
- 이를 통해  $\{\mathbf{u}_1, \dots, \mathbf{u}_k\}, \{\mathbf{v}_1, \dots, \mathbf{v}_k\}$ 를 얻을 수 있음.
- 하지만  $U \in \mathbb{C}^{m \times m}, V^T \in \mathbb{C}^{n \times n}$  인 unitary matrix로 확장해야 함.  
→ 따라서 U에는  $(m - k)$ 개, V에는  $(n - k)$ 개만큼의 arbitrary orthonormal columns를 추가해야 함.
- $\Sigma$ 는 주 대각성분 외에 0으로 채워 넣기

따라서

- $\{\mathbf{u}_1, \dots, \mathbf{u}_k, \mathbf{u}_{k+1}, \dots, \mathbf{u}_m\}$ 은  $\{\mathbf{u}_1, \dots, \mathbf{u}_k\}$ 로부터  $R^m$ 의 정규직교기저로 확장한 것이다.
- $\{\mathbf{v}_1, \dots, \mathbf{v}_k, \mathbf{v}_{k+1}, \dots, \mathbf{v}_n\}$ 은  $\{\mathbf{v}_1, \dots, \mathbf{v}_k\}$ 로부터  $R^n$ 의 정규직교기저로 확장한 것이다.



# Singular Value Decomposition

## Full SVD and Reduced SVD

· Full SVD의 형태

i)  $n < m$

$$A = \begin{array}{c|c|c} \text{blue box} & \text{purple box} & \text{white box with diagonal} \\ \hline A & U & \Sigma \\ & & V^H \end{array}$$

ii)  $m < n$

$$A = \begin{array}{c|c|c} \text{blue box} & \text{purple box} & \text{white box with diagonal} \\ \hline A & U & \Sigma \\ & & V^H \end{array}$$

▲ Figure 4

· Reduced SVD의 형태

i)  $n < m$

$$A = \begin{array}{c|c|c} \text{blue box} & \text{purple box} & \text{white box with diagonal} \\ \hline A & \hat{U} & \hat{\Sigma} \hat{V}^H \end{array}$$

ii)  $m < n$

$$A = \begin{array}{c|c|c} \text{blue box} & \text{purple box} & \text{white box with diagonal} \\ \hline A & \hat{U} & \hat{\Sigma} \hat{V}^H \end{array}$$

▲ Figure 5

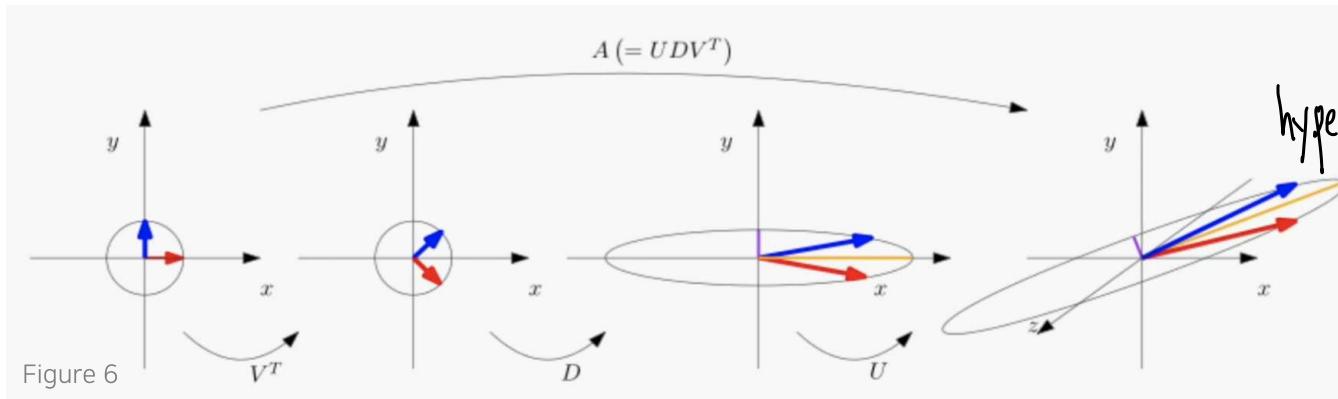
$\Sigma$ 의 zero rows 또는 zero columns인 부분들은 생략해서 쓸 수 있다!  
→ 사라지는 벡터들을 생략해 작성하자 → Reduced SVD

이때,  $U \in \mathbb{C}^{m \times k}$ ,  $V^T \in \mathbb{C}^{k \times n}$ 로 축소

# Singular Value Decomposition

## SVD의 차원축소 과정

$$\begin{pmatrix} 2 & 2 \\ -\frac{1}{2} & \frac{1}{2} \\ -2 & -2 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ 0 & 1 & 0 \\ -\frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} 4 & 0 \\ 0 & \frac{1}{\sqrt{2}} \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix} \quad \rightarrow \quad A = U\Sigma V^T$$

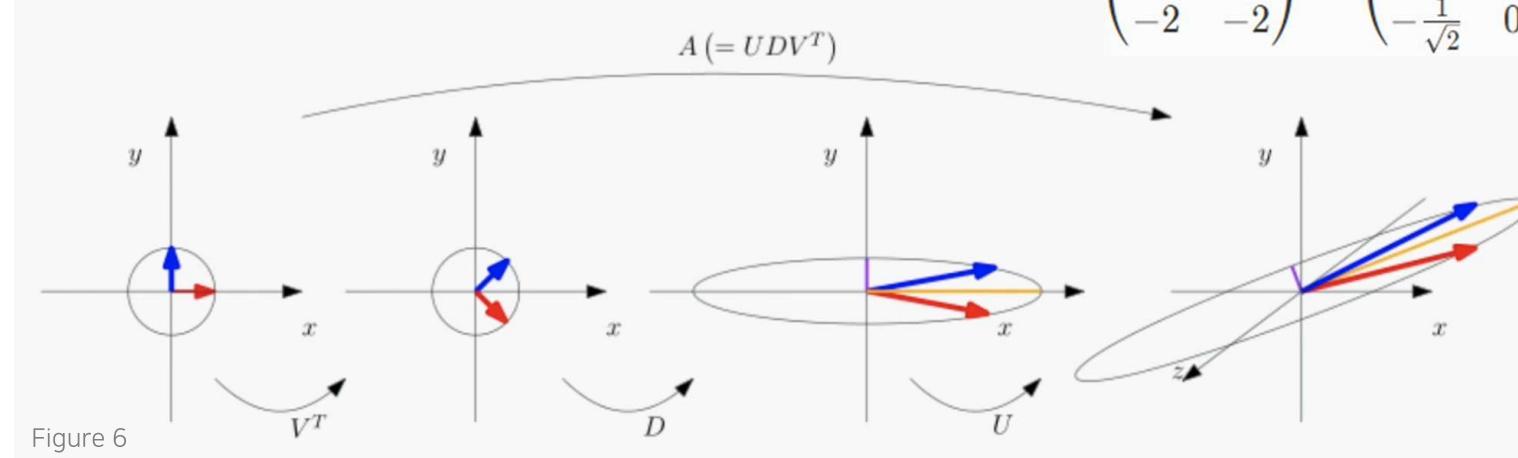


그림에서 주목해야 할 점은, SVD가 행렬이 벡터에 대해 선형변환을 하는 과정을 설명해준다는 것!

- 이 단위 원 위의 벡터들은  $V^T$ ,  $\Sigma$ ,  $U$ 에 의해 rotation – extension – rotation 과정을 거쳐 hyperellipse로 mapping 된다.
- 양변에  $2 \times 2$  identity matrix를 곱하는 상황을 생각 → 그림의 빨강, 파랑벡터는 각각  $(1,0)'$ ,  $(0,1)'$ 를 의미한다.
- 즉,  $A$ 로 인해  $(1,0)', (0,1)'$ 가  $\begin{pmatrix} 2, -\frac{1}{2}, -2 \end{pmatrix}', \begin{pmatrix} 2, \frac{1}{2}, -2 \end{pmatrix}'$ 로 mapping 되는 과정을 알 수 있다!

# Singular Value Decomposition

## SVD의 차원축소 과정



$$\begin{pmatrix} 2 & 2 \\ -\frac{1}{2} & -2 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 4 & 0 \\ 0 & \frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}$$

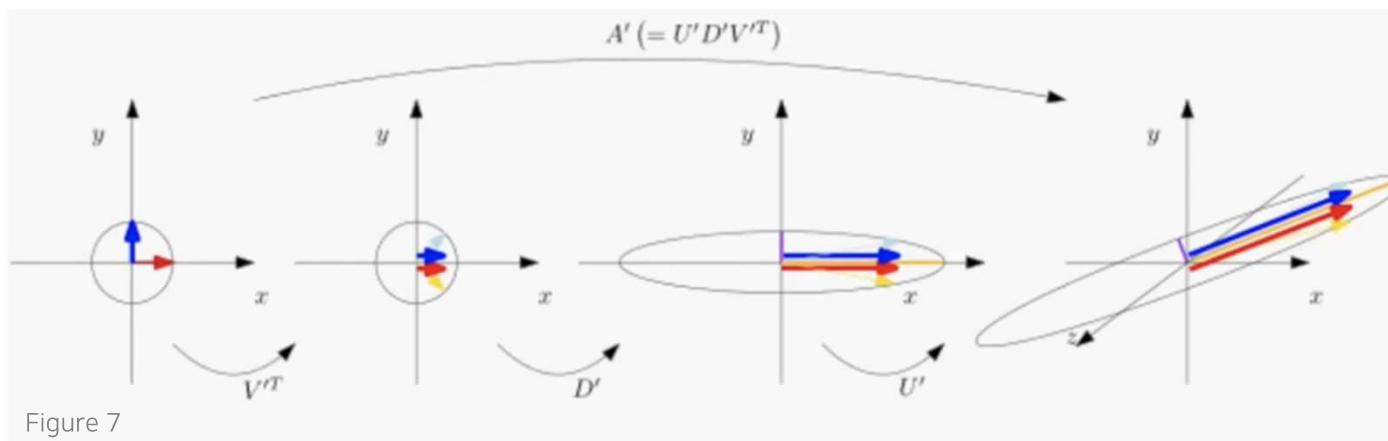
- 직관적으로 mapping된 빨강, 파랑 벡터의 정보 손실을 최소화하면서 잘 요약해주는 벡터 방향은 두 벡터의 평균치임.
- 이 평균치는  $(2, 0, -2)'$ 이며 이를 정규화 한 것은  $u_1$ 과 일치한다!
- 따라서 데이터의 feature 수를 줄이고 싶다면, singular value가 가장 큰 4에 대응하는 left singular vector인  $u_1$ 으로 projection해야 정보손실을 최소화 할 수 있다.

# Singular Value Decomposition

## SVD의 차원축소 과정

$u_1$ 을 이용해, 즉 1개의 singular value를 이용해 A를 압축한 결과는 다음과 같다.

$$\begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ -\frac{1}{\sqrt{2}} \end{pmatrix} (4) \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} 2 & 2 \\ 0 & 0 \\ -2 & -2 \end{pmatrix} \quad \rightarrow \quad U'_{3 \times 1} \Sigma'_{1 \times 1} V'^T_{1 \times 2} = A'_{3 \times 2}$$



- 따라서 가장 큰 singular value 한 개를 통해  $A'$ 을 만들 수 있으며, 빨강, 파랑 벡터들이 해당 방향으로 동일하게 mapping 되는 것 확인할 수 있다.
- 단위 원 안의 어느 벡터를 행렬  $A$ 를 통해 mapping한 결과는  $u_1$  방향으로 정보가 압축된다.

# Singular Value Decomposition

## Low Rank Approximation

이 개념을 확장한 것이 바로 low rank approximation, 이는 rank  $k$ 인  $A$ 의 Reduced SVD를 풀어서 작성한 것이다.

$$A = [\mathbf{u}_1 \ \mathbf{u}_2 \ \cdots \ \mathbf{u}_k] \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_k \end{bmatrix} \begin{bmatrix} \mathbf{v}_1^T \\ \mathbf{v}_2^T \\ \vdots \\ \mathbf{v}_k^T \end{bmatrix} \quad \rightarrow \quad A = \sigma_1 \mathbf{u}_1 \mathbf{v}_1' + \sigma_2 \mathbf{u}_2 \mathbf{v}_2' + \cdots + \sigma_k \mathbf{u}_k \mathbf{v}_k'$$

Figure 8

▶ Thm1. 어떤  $0 \leq v \leq k$ 인  $v$ 에 대해,  $A_v = \sum_{j=1}^v a_j u_j v_j^T$  라고 정의하자.

그러면,  $\|A - A_v\|_2 = \inf \|A - B\|_2 = \sigma_{v+1}$  이다. ( 이 때,  $B \in \mathbb{C}^{m \times n}, \text{rank}(B) \leq v$  )

이 때  $v = p = \min\{m, n\}$  이면,  $\sigma_{v+1} = 0$

▶ Thm2. 어떤  $0 \leq v \leq k$ 인  $v$ 에 대해, 위에서 정의한  $A_v$ 는

$\|A - A_v\|_F = \inf \|A - B\|_F = \sqrt{\sigma_{v+1}^2 + \cdots + \sigma_k^2}$  을 만족한다. (이 때  $B \in \mathbb{C}^{m \times n}, \text{rank}(B) \leq v$  )



# Singular Value Decomposition

## Low Rank Approximation

▶ Thm1. 어떤  $0 \leq v \leq k$ 인  $v$ 에 대해,  $A_v = \sum_{j=1}^v a_j u_j v_j^T$  라고 정의하자.

그러면,  $\|A - A_v\|_2 = \inf \|A - B\|_2 = \sigma_{v+1}$  이다. ( 이 때,  $B \in \mathbb{C}^{m \times n}, \text{rank}(B) \leq v$  )

이 때  $v = p = \min\{m, n\}$  이면,  $\sigma_{v+1} = 0$

▶ Thm2. 어떤  $0 \leq v \leq k$ 인  $v$ 에 대해, 위에서 정의한  $A_v$ 는

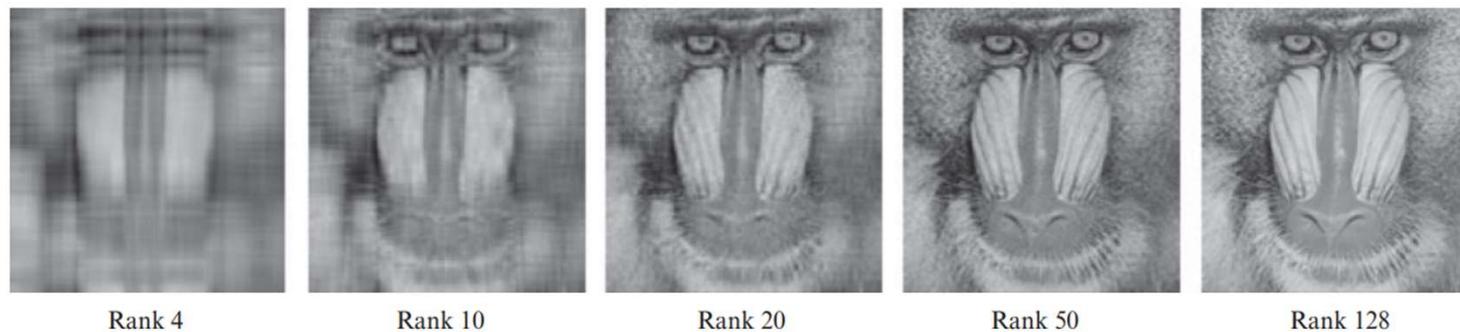
$\|A - A_v\|_F = \inf \|A - B\|_F = \sqrt{\sigma_{v+1}^2 + \dots + \sigma_k^2}$  을 만족한다. ( 이 때,  $B \in \mathbb{C}^{m \times n}, \text{rank}(B) \leq v$  )

- 이 때, singular value는 내림차순으로 정렬되므로  $\sigma_{v+1}$ 를 상당히 작게 할 수 있다.
- 이는  $A'$ 과  $A$ 의 norm 차이가 상당히 작을 수 있다는 것을 암시한다.
- 따라서, 우리는 모든 singular value 를 쓰지 않아도 low rank approximation을 통해  $A$ 를  $A'$ 으로 충분히 근사 가능하다!

# Singular Value Decomposition

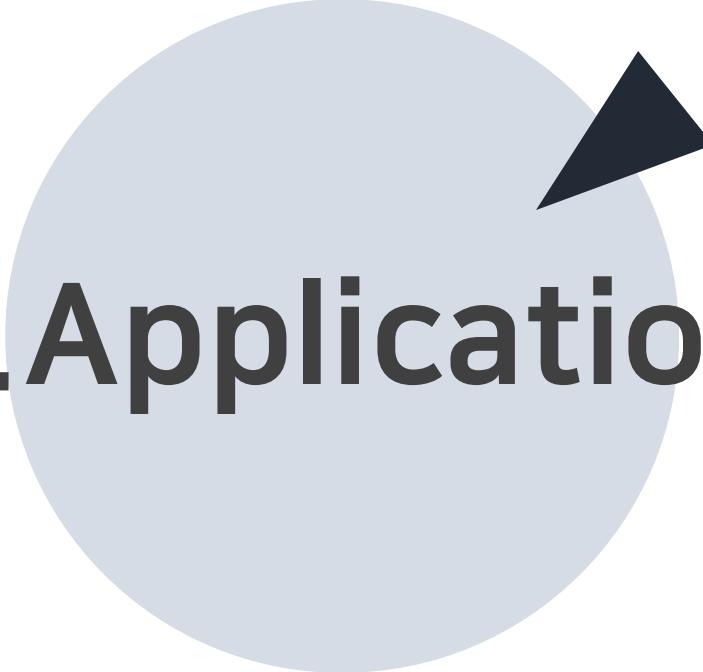
## Low Rank Approximation

Ex)  $1000 \times 1000$ 이고 rank가 200인 원숭이 그림



▲ Figure 9

- 원본 이미지를 저장한다면  $m \times n = 1,000 \times 1,000 = 1,000,000$ 의 저장공간이 필요 (상당히 비효율적)
- rank가 200이므로 reduced SVD를 사용하면  $U\Sigma V^T$ 에서  $U$ 는  $1000 \times 200$ ,  $V^T$ 는  $200 \times 1000$ ,  $\Sigma$ 는 200개의 singular values로 이루어진다.  
그러므로  $k(m + n + 1) = 200(1,000 + 1,000 + 1) = 400,200$ 개의 저장 공간만 필요
- 추가적으로 low rank approximation을 이용한다면 ( $r$ 개의 singular value를 선택)  
 $r(m + n + 1)$ 개의 저장공간만 활용 -> 상당히 효율적이다! [1]



### 3. Application

# Application

## Matrix Properties via the SVD

$A$ 는  $m \times n$  차원의 행렬,  $r$ 은  $A$ 의 rank,  $p = \min(m, n)$ 는  $A$ 의 0이 아닌 singular value들의 개수( $r \leq p$ )라 할 때

- Thm1.  $A$ 의 rank는  $r$ , 즉 0이 아닌 singular value들의 개수이다
- Thm2,  $\text{range}(A) = \text{span}(u_1, u_2, \dots, u_r)$ 이고,  $\text{null}(A) = \text{span}(v_{r+1}, v_{r+2}, \dots, v_n)$ 이다
- Thm3.  $\|A\|_2 = \sigma_1$ 이고,  $\|A\|_F = \sqrt{\sigma_1^2 + \sigma_2^2 + \dots + \sigma_r^2}$ 이다  $\|A\|_2 = \|V\Sigma V^\top\|_2 = \|\Sigma\|_2 = \sigma_{\max}$
- Thm4.  $\text{cond}_2(A) = \sigma_{\max}/\sigma_{\min}$

$$\|A\|_2 = \max_{\mathbf{z}} \frac{\|A\mathbf{z}\|}{\|\mathbf{z}\|}$$

# Application

## Pseudoinverse(유사 역행렬)

1. 0이 아닌 스칼라의 pseudoinverse는 그것의 역수이다. 0의 pseudoinverse는 그대로 0이다.
2.  $m \times n$  대각 행렬의 pseudoinverse는 각 원소(스칼라)에 pseudoinverse를 취하고 전체를 transpose해서 구한다

$$A = \begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \\ 0 & 0 & 0 \end{pmatrix} \quad A^\dagger = \begin{pmatrix} 1/a & 0 & 0 & 0 \\ 0 & 1/b & 0 & 0 \\ 0 & 0 & 1/c & 0 \end{pmatrix}$$

3. 역행렬이 존재하는 행렬의 pseudoinverse는 그것의 역행렬과 같다

$$A^\dagger = A^{-1}$$

4. 일반적인 행렬의 pseudoinverse는 다음과 같다.

$$A^\dagger = (U\Sigma V^T)^\dagger = V\Sigma^\dagger U^T$$



# Application

## Pseudoinverse를 통해 Least Square Solution 구하기

$Ax \approx b$

$$x_{LS} = (A^T A)^{-1} A^T b = A^\dagger b$$

증명:

$$\begin{aligned} \|\mathbf{r}(x)\|_2^2 &= \|Ax - b\|_2^2 = \|U^T(U\Sigma V^T x - b)\|_2^2 = \|\Sigma V^T x - U^T b\|_2^2 = \\ &\| \begin{bmatrix} S & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} V_L^T \\ V_R^T \end{bmatrix} x - \begin{bmatrix} U_L^T \\ U_R^T \end{bmatrix} b \|_2^2 = \|SV_L^T x - U_L^T b\|_2^2 + \|U_R^T b\|_2^2 \end{aligned}$$

$$SV_L^T x - U_L^T b = 0 \rightarrow x = V_L S^{-1} U_L^T b$$

이때,  $V_L S^{-1} U_L^T$ 는  $V\Sigma^\dagger U^T$ 의 Reduced SVD이므로,  $V_L S^{-1} U_L^T = V\Sigma^\dagger U^T = A^\dagger$

$$\text{따라서 } x_{LS} = A^\dagger b$$

참고:

$$A^\dagger = V\Sigma^\dagger U^T$$

$$\begin{aligned} \vdots \quad A &= U\Sigma V^T = \begin{bmatrix} U_L & U_R \end{bmatrix} \begin{bmatrix} S & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} V_L & V_R \end{bmatrix}^T \end{aligned}$$



# Application

## Conditioning & Pseudoinverse

$$A = \begin{bmatrix} 0.913 & 0.659 \\ 0.780 & 0.563 \\ 0.457 & 0.330 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} 1.572 \\ 1.343 \\ 0.787 \end{bmatrix}, \tilde{\mathbf{b}} = \begin{bmatrix} 1.57 \\ 1.34 \\ 0.78 \end{bmatrix}$$

$$\hat{A} = \begin{bmatrix} 0.91298 & 0.65902 \\ 0.77999 & 0.56302 \\ 0.45706 & 0.32992 \end{bmatrix}$$

$$A^\dagger = \begin{bmatrix} 1417.326 & 1255.577 & -4972.410 \\ -1962.757 & -1738.785 & 6888.017 \end{bmatrix}$$

$$\hat{A}^\dagger = \begin{bmatrix} 0.364 & 0.311 & 0.182 \\ 0.262 & 0.224 & 0.131 \end{bmatrix}$$

$$A^\dagger \mathbf{b} = \begin{bmatrix} 0.99 \\ 1.01 \end{bmatrix} A^\dagger \tilde{\mathbf{b}} = \begin{bmatrix} 29.19 \\ -38.07 \end{bmatrix}$$

$$\hat{A}^\dagger \mathbf{b} = \begin{bmatrix} 1.132 \\ 0.817 \end{bmatrix} \hat{A}^\dagger \tilde{\mathbf{b}} = \begin{bmatrix} 1.129 \\ 0.815 \end{bmatrix}$$



# Application

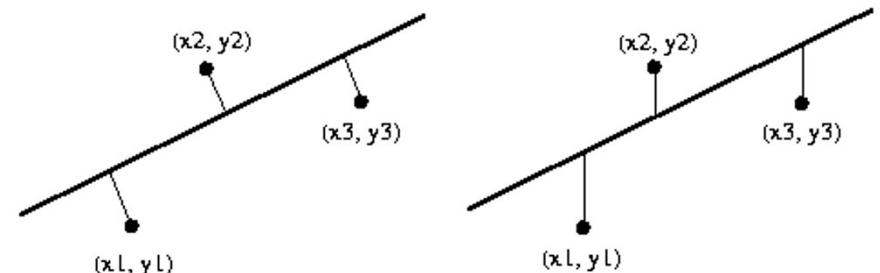
## Total Least Square(TLS)

### Ordinary Least Square(OLS)

- $Ax \cong b$ 를 풀 때,  $A$ 는 변경하지 않고  $b$ 만  $proj_{col(A)} b$ 로 바꿔서 solution을 구한다
- Data point와 curve 간의 vertical distance를 최소화한다

### Total Least Square(TLS)

- $B$ 뿐만 아니라 행렬  $A$ 도 바꿔서 solution을 구한다
- Data point와 curve 간의 orthogonal(perpendicular) distance를 최소화한다



Perpendicular Distances

▲ Figure 10

Vertical Distances

# Application

## Solution of Total Least Square

- 행렬  $A$ 를 rank가  $n$ 인  $m \times n$  행렬이라고 하자
- 그러면  $[A \ b]$ 의 rank는  $n+1$ 일 것이다( $b$ 가  $\text{Col}(A)$  안에 있지 않다면)
- $[A \ b]$ 를 SVD 한 후 lowest singular value를 0으로 만들어 rank를  $n$ 으로 만들어 준다
- 그렇게 만들어진 행렬을  $[\hat{A} \ y]$ 라 하자
- $\hat{A}x = y \rightarrow [\hat{A} \ y] \begin{bmatrix} x \\ -1 \end{bmatrix} = 0$
- $\begin{bmatrix} x \\ -1 \end{bmatrix}$ 는  $[\hat{A} \ y]$ 의 null space 안에 있음
- $[\hat{A} \ y]$ 의 null space는  $\text{span}(v_{n+1})$ 이다
- 따라서  $\begin{bmatrix} x \\ -1 \end{bmatrix}$ 는  $\text{span}(v_{n+1})$ 안에 있고,  $\begin{bmatrix} x \\ -1 \end{bmatrix}$ 는  $v_{n+1}$ 선형 결합으로 표현이 가능하다.



# Application

## Solution of Total Least Square

o 
$$\begin{bmatrix} x_1 \\ \vdots \\ x_n \\ -1 \end{bmatrix} = a \begin{bmatrix} v_{1,n+1} \\ \vdots \\ v_{n,n+1} \\ v_{n+1,n+1} \end{bmatrix} \rightarrow \text{마지막 원소가 } -10 \text{이 되어야 하므로 } a = -\frac{1}{v_{n+1,n+1}}$$

o 
$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = -\frac{1}{v_{n+1,n+1}} \begin{bmatrix} v_{1,n+1} \\ \vdots \\ v_{n,n+1} \end{bmatrix}$$

# Application

## TLS 예제

다음의 data point들이 있을 때 linear model  $f(t, x) = xt$ 의 기울기  $x$ 를 찾고자 한다

$t$	-2	-1	3
$y$	-1	3	-2

$$\hat{x} = -\frac{1}{\sqrt{v_{n+1, n+1}}} \begin{bmatrix} v_{1, n+1} \\ \vdots \\ v_{n, n+1} \end{bmatrix}$$

풀이:

$$\begin{bmatrix} -2 \\ -1 \\ 3 \end{bmatrix} x = \begin{bmatrix} -1 \\ 3 \\ -2 \end{bmatrix}$$

$$[\mathbf{A} \quad \mathbf{b}] = [\mathbf{t} \quad \mathbf{y}] = \begin{bmatrix} -2 & -1 \\ -1 & 3 \\ 3 & -2 \end{bmatrix} =$$

$$\begin{bmatrix} -0.154 & 0.802 & 0.577 \\ -0.617 & -0.535 & 0.577 \\ 0.772 & -0.267 & 0.577 \end{bmatrix} \begin{bmatrix} 4.583 & 0 \\ 0 & 2.646 \end{bmatrix} \begin{bmatrix} 0.707 & -0.707 \\ -0.707 & -0.707 \end{bmatrix} = \mathbf{U} \Sigma \mathbf{V}^T$$

$$v_{n+1} = v_2 = \begin{bmatrix} -0.707 \\ -0.707 \end{bmatrix}$$
$$x = -\frac{1}{-0.707} - 0.707 = -1$$

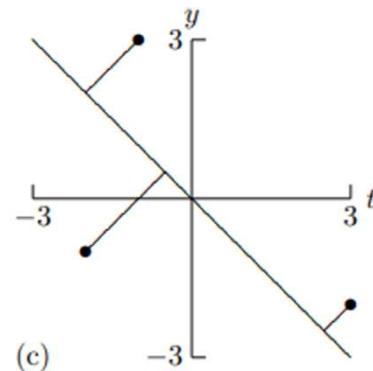


# Application

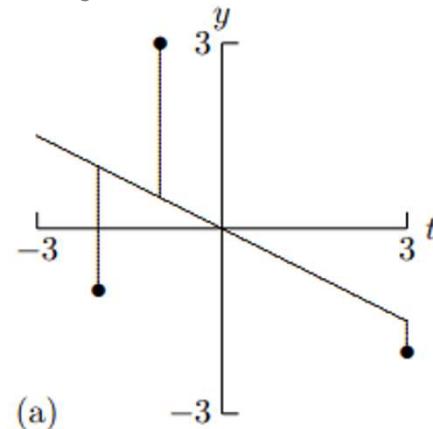
## TLS 예제

TLS는 data point와 curve 간의 orthogonal distance를 최소화하므로  
data point들과 curve를 좌표평면에 표현하면 다음과 같이 나타난다

비교) OLS의 solution



▲ Figure 11



▲ Figure 12

# Application

## Eigenvalue Decomposition

$X \in \mathbb{C}^{m \times m}$ 의 열벡터들이 matrix  $A \in \mathbb{C}^{m \times m}$ 의 선형 독립인 eigenvectors로 구성될 때,  
 $A$ 의 eigenvalue decomposition은  $A = X\Lambda X^{-1}$ 이다. 이때  $\Lambda$ 는 대각성분이  $A$ 의  
eigenvalues로 구성된  $m \times m$  diagonal matrix이다.

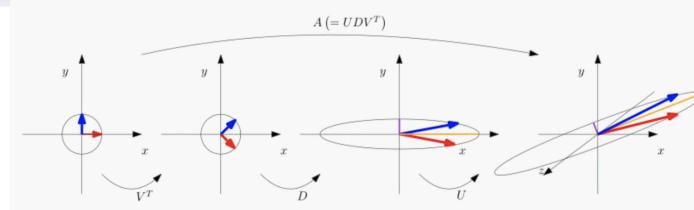


# Application

## SVD vs Eigenvalue Decomposition

$A = U\Sigma V^T$	$A = X\Lambda X^{-1}$
두 개의 기저 사용 $[T]_{B,B} = [I]_{B,U}[T]_{U,V}[I]_{V,B}$	한 개의 기저 사용 $[T]_{B,B} = [I]_{B,X}[T]_{X,X}[I]_{X,B}$
Orthonormal basis	Linearly independent한 eigenvector들로 구성된 basis(일반적으로 orthonormal하지 않음)
모든 matrix에 적용 가능 (rectangular한 모양일 때도)	어떤 matrix에는 적용 불가능 (square한 모양일지라도)
A 자체에 관한 문제일 때 주로 사용	A의 iterated form에 관한 문제일 때 주로 사용 Ex) $A^k, e^{tA}$

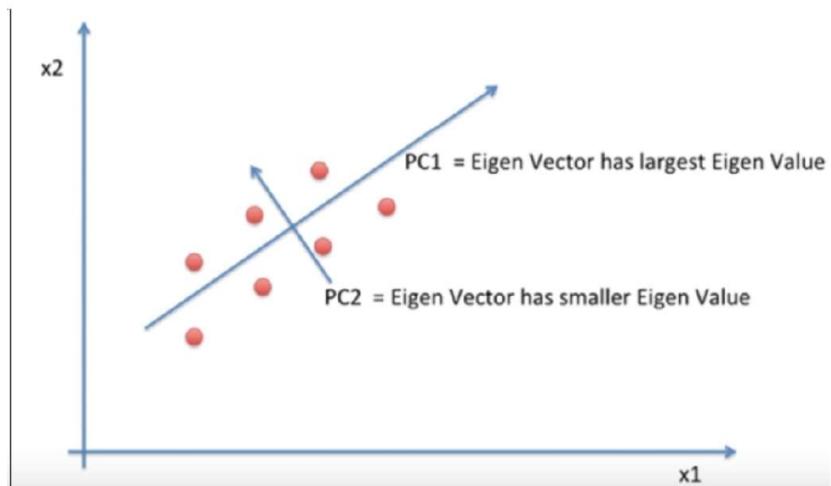
$$\begin{pmatrix} 2 & 2 \\ -\frac{1}{2} & \frac{1}{2} \\ -2 & -2 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ 0 & 1 & 0 \\ -\frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} 4 & 0 \\ 0 & \frac{1}{\sqrt{2}} \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}$$



# Application

## PCA

- 차원 축소에 사용되는 대표적인 알고리즘으로, SVD가 사용되는 예시 중 하나
- 계산 비용이 많고, 분석에 필요한 시각화가 어려운 고차원 데이터를 계산 비용이 비교적 작고 준수한 성능의 저차원 데이터로!
- 원 데이터의 표현력을 잘 보존하기 위해 분산을 최대로 하는 주축을 찾는다. 이때 주축은 표본들의 공분산 행렬의 eigenvector들



▲ Figure 13

# Application

## PCA에서 주축이 공분산 행렬의 eigenvector들이 되는 수식적 증명

- $\mathbf{z}_i = (x_1, \dots, x_p), i = 1, \dots, n, \|\mathbf{w}\| = 1, \mathbf{h}_i = (\mathbf{z}_i \cdot \mathbf{w})\mathbf{w}$
- $(\mathbf{z}_i \cdot \mathbf{w})$ 의 분산을 최대화하는  $\mathbf{w}$ 를 찾기
- $(\mathbf{z}_i \cdot \mathbf{w})$ 들의 분산:  $\sigma_{\mathbf{w}}^2, Z = \begin{bmatrix} \mathbf{z}_1^T \\ \vdots \\ \mathbf{z}_n^T \end{bmatrix}$
- $\sigma_{\mathbf{w}}^2 = (1/n) \sum_i (\vec{z}_i \cdot \vec{w})^2 - ((1/n) \sum_i (\vec{z}_i \cdot \vec{w}))^2 = (1/n) \sum_i (\vec{z}_i \cdot \vec{w})^2 = (1/n)(Z\vec{w})^T (Z\vec{w}) = (1/n)\vec{w}^T Z^T Z \vec{w} = \vec{w}^T ((Z^T Z)/n) \vec{w} = \vec{w}^T C \vec{w}$
- 제약조건  $\mathbf{w}^T \mathbf{w} = 1$ 에서  $\mathbf{w}^T C \mathbf{w}$ 의 최대값을 찾는 문제  $\rightarrow$  라그랑주 승수법
- $u = \vec{w}^T C \vec{w} - \lambda(\vec{w}^T \vec{w} - 1)$
- $u$ 를 최대로 하는  $\mathbf{w}$ 는  $\frac{\partial u}{\partial \mathbf{w}} = 0$ 이 되게 하는 값, 즉  $\frac{\partial u}{\partial \mathbf{w}} = 2C\vec{w} - 2\lambda\vec{w} = 0$   
 $C\vec{w} = \lambda\vec{w}$



# Application

## SVD 사용하는 이유, python code

공분산행렬은 대칭행렬, 즉

$$A = U\Sigma V^T = PDP^T$$

Eigenvalue decomposition 대신 SVD를 사용하는 이유? scikit-learn에서 SVD의

계산 속도가 eigenvalue decomposition의 계산 속도보다 빠름!

```
import pandas as pd
import matplotlib.pyplot as plt
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
df = pd.read_csv(url, names=['sepal length','sepal width','petal length','petal width','target'])
print(df)
   sepal length  sepal width  petal length  petal width     target
0          5.1         3.5         1.4         0.2    Iris-setosa
1          4.9         3.0         1.4         0.2    Iris-setosa
2          4.7         3.2         1.3         0.2    Iris-setosa
3          4.6         3.1         1.5         0.2    Iris-setosa
4          5.0         3.6         1.4         0.2    Iris-setosa
..          ...
145         6.7         3.0         5.2         2.3  Iris-virginica
146         6.3         2.5         5.0         1.9  Iris-virginica
147         6.5         3.0         5.2         2.0  Iris-virginica
148         6.2         3.4         5.4         2.3  Iris-virginica
149         5.9         3.0         5.1         1.8  Iris-virginica
```

▲ Figure 15

```
from sklearn.preprocessing import StandardScaler # 표준화 패키지 라이브러리
x = df.drop(['target'], axis=1).values # 독립변인들의 value값만 추출
y = df['target'].values # 종속변인 추출
x = StandardScaler().fit_transform(x) # x객체에 x를 표준화한 데이터를 저장
features = ['sepal length', 'sepal width', 'petal length', 'petal width']
pd.DataFrame(x, columns=features).head()
print(pd.DataFrame(x, columns=features))

   sepal length  sepal width  petal length  petal width
0      -0.900681     1.032057    -1.341272    -1.312977
1      -1.143017    -0.124958    -1.341272    -1.312977
2      -1.385353     0.337848    -1.398138    -1.312977
3      -1.506521     0.106445    -1.284407    -1.312977
4      -1.021849     1.263460    -1.341272    -1.312977
..      ...
145     1.038005    -0.124958     0.819624     1.447956
146     0.553333    -1.281972     0.705893     0.922064
147     0.795669    -0.124958     0.819624     1.053537
148     0.432165     0.800654     0.933356     1.447956
149     0.068662    -0.124958     0.762759     0.790591
```

[150 rows x 4 columns]

▲ Figure 16



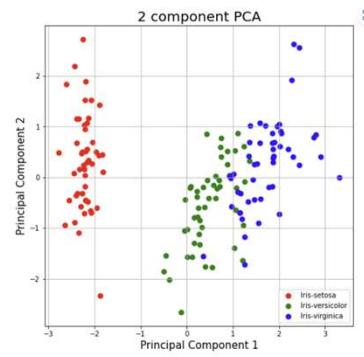
# Application

## python code

```
from sklearn.decomposition import PCA
#3번 4번 성분은 0.04정도로 설명 가능한 분산량이 얼마 증가하지 않기 때문에,
#주성분은 2개로 결정하는 것이 적절하다.
#점점 작아지도록 정렬되어 있음을 알 수 있다.
pca = PCA(n_components=4)
print(pca.explained_variance_ratio_)
[0.72770452 0.23030523 0.03683832 0.00515193]

pca = PCA(n_components=2) # 주성분을 몇개로 할지 결정
print(pca.explained_variance_ratio_)
[0.72770452 0.23030523]
print(pca.components_)
principal_components = pca.fit_transform(x)
principalDF = pd.DataFrame(data=principalComponents, columns = ['principal component1', 'principal component2', '3', '4'])
print(pca.explained_variance_ratio_)
[0.72770452 0.23030523]
print(principal_components)
principal component1 principal component2
0 -2.264542 0.505704
1 -2.088426 -0.655405
2 -2.367950 -0.318477
3 -2.304197 -0.575368
4 -2.388777 0.674767
.. ...
145 1.870522 0.382822
146 1.558492 -0.905314
147 1.520845 0.266795
148 1.376391 1.016362
149 0.959299 -0.022284
[150 rows x 2 columns]
```

▲ Figure 17



▲ Figure 18

**END**

# References

[1] Howard Anton, Chris Rorres - Elementary Linear Algebra, Applications Version-Wiley (2013), 523p

## Figure

[1] <https://github.com/Hoon34>

[2 – 3, 8, 9] Howard Anton, Chris Rorres - Elementary Linear Algebra, Applications Version-Wiley (2013)

[4, 5] Darve and Wootters - Numerical Linear Algebra with Julia

[6, 7] <https://velog.io/@dltjrdud37/Day9-벡터와-직교분해-SVD-PC>

[10] <https://services.math.duke.edu/education/modules2/materials/test/test/part4.html>

[11, 12] Michael T. Heath - Scientific Computing

[13, 15, 16, 17, 18] <https://velog.io/@swan9405/PCA>

[14] <https://darkpgmr.tistory.com/m/110>

