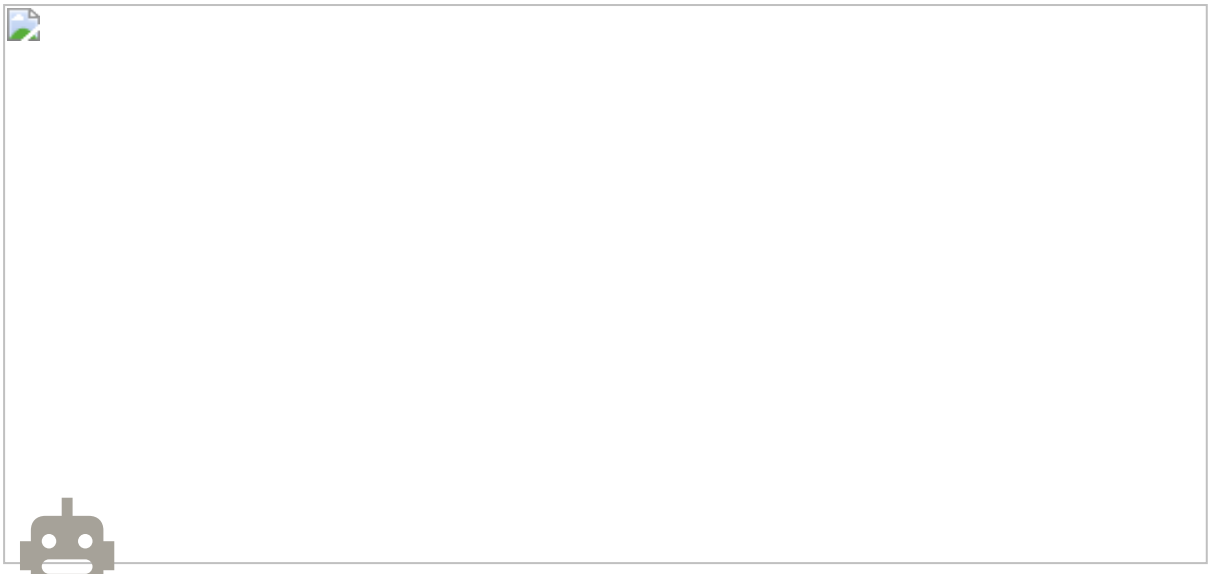


6장. 차원 축소

💡 매우 많은 피처로 구성된 다차원 데이터 세트의 차원을 축소해 새로운 차원의 데이터 세트를 생성하는 것

- 다차원 데이터 세트의 문제점
  1. 차원이 증가할 수록 데이터 포인트 간의 거리가 기하급수적으로 멀어지고, 희소한 Sparse 구조를 가져 예측 신뢰도가 떨어진다.
  2. 다중공선성 문제(독립변수 간의 상관관계가 높은 것)로 예측 성능 저하
     - 회귀분석의 전제 가정 위배 : 독립변수간 상관관계는 높으면 안된다
- 차원 축소의 분류
  - 피처(특성) 선택 : 특정 피처에 종속성이 강한 불필요 피처는 아예 제거 + 데이터 특징 잘 나타내는 주요 피처만 선택
  - 피처(특성) 추출 : 기존 피처를 저차원의 중요 피처로 압축하여 추출 ⇒ 기존 피처와 완전히 다른 새로운 값이 됨
    - 단순 압축이 아닌, 피처를 합축적으로 더 잘 설명할 수 있는 또 다른 공간으로 매칭하여 추출하는 것ex) 학생의 모의고사성적, 내신성적, 수능성적, 봉사활동, 대외활동, 수상경력 등 ⇒ 학업 성취도, 커뮤니케이션 능력, 문제해결력 등 더 함축적인 요약 특성으로 추출할 수 있음

    - 가장 중요한 의미 : 데이터를 더 잘 설명할 수 있음 잠재적인 요소 추출 : **PCA, SVD, NMF** 등

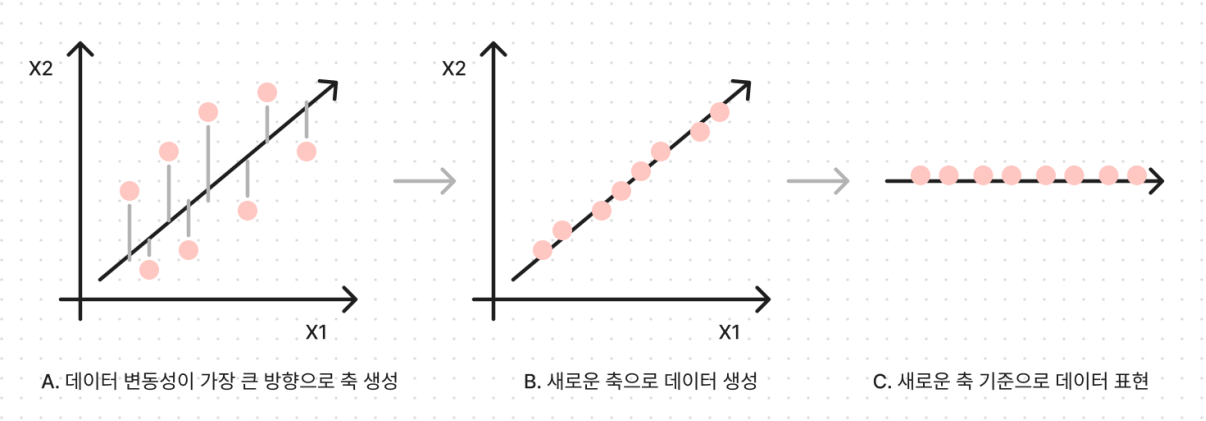
- 차원 축소의 활용
    1. 이미지 데이터에서 잠재된 특성을 피처로 도출해 함축적 형태의 이미지 변환과 압축 수행 ⇒ 원본보다 작은 차원으로 과적향 방지
    2. 텍스트 문서의 숨겨진 의미 추출. 문서 내 단어들의 구성에서 숨겨져 있는 시맨틱 Semantic 의미나 토픽topic을 잠재 요소로 간주하고 이를 찾아낸다.

1️⃣ PCA(Principal Component Analysis) : 주성분 분석

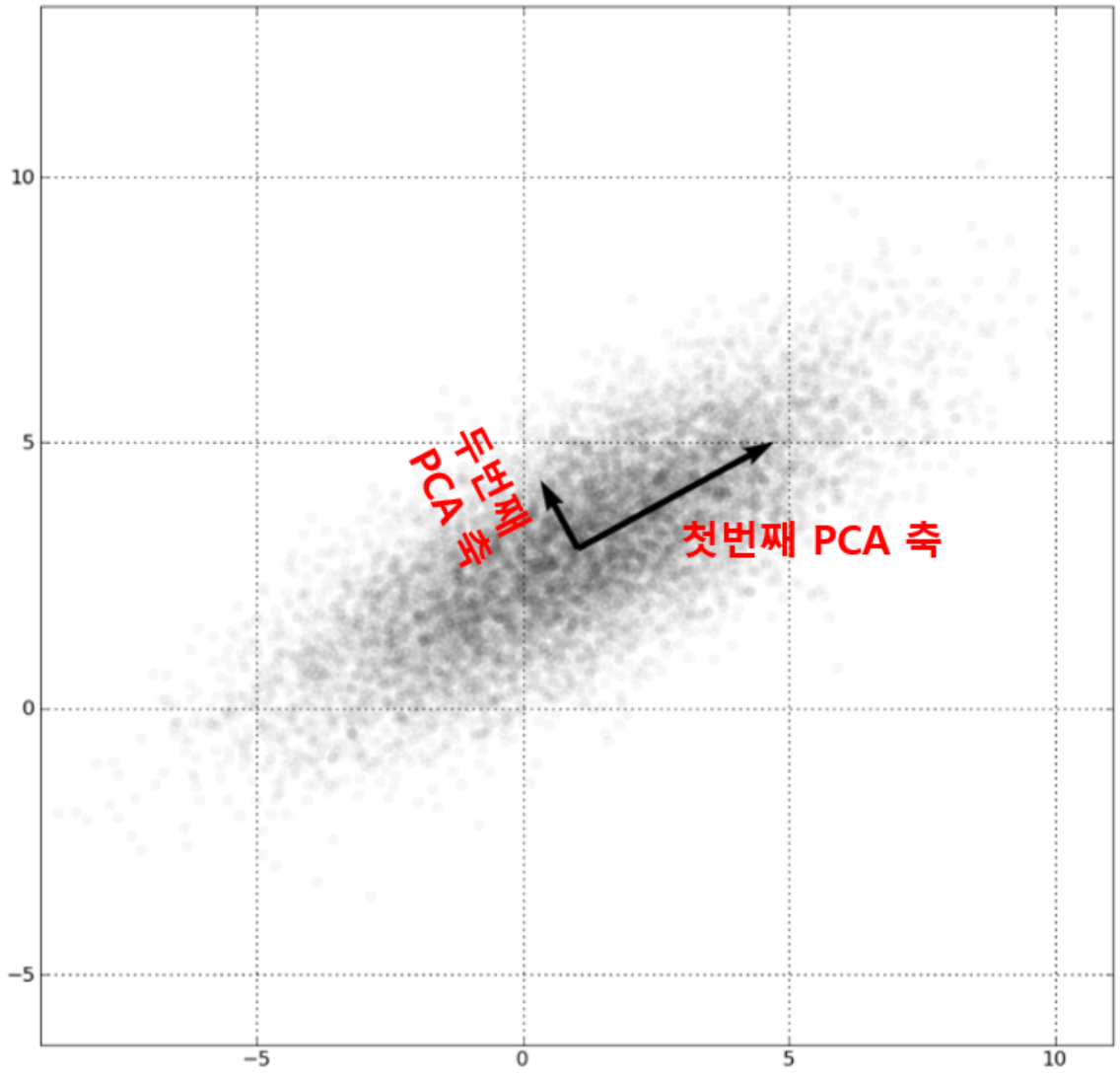
1-1. PCA 개요

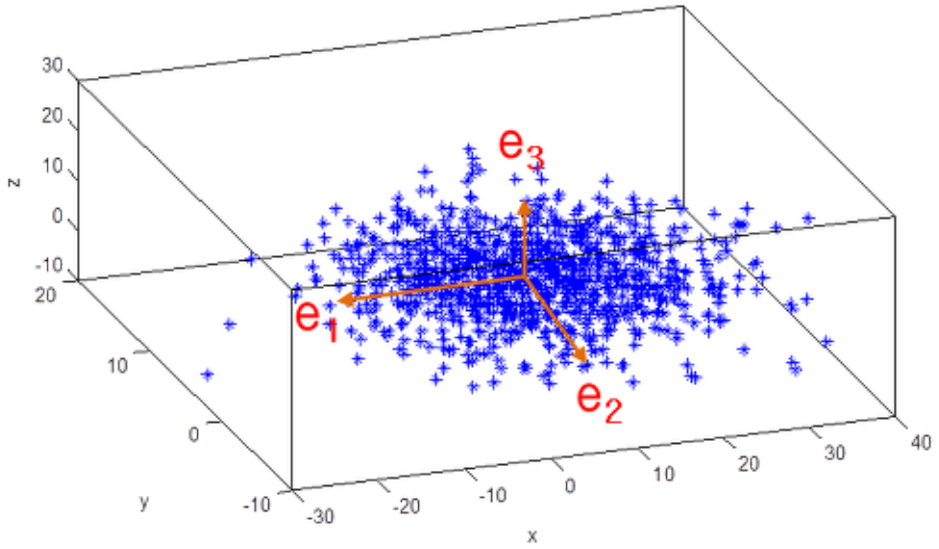
여러 변수 간에 존재하는 상관관계를 이용해 이를 대표하는 주성분을 추출해 차원을 축소하는 기법

- PCA의 주성분: 정보 유실을 최소화하기 위해 가장 높은 분산을 가지는 데이터를 찾아, 이 축으로 차원을 축소한다. 즉, 분산이 데이터의 특성을 가장 잘 나타내는 것으로 간주한다.
- PCA 차원 축소 하는 방법



- 첫 번째 벡터 축 : 가장 큰 데이터 변동성(Variance)을 기반으로 생성
- 두 번째 벡터 축 : 첫 번째 벡터 축에 직각이 되는 벡터(직교 벡터)를 축으로 함
- 세 번째 벡터 축 : 다시 두 번째 축과 직각이 되는 벡터를 설정하는 방식으로 축 생성
    - 선형대수 관점 : 입력 데이터의 공분산 행렬(Covariance Matrix)을 고유값 분해하고, 이렇게 구한 고유벡터에 입력 데이터를 선형 변환하는 것


두번째 PCA 축
첫번째 PCA 축
10
5
0
−5
−5
0
5
10


e3
e1
e2
z
y
x

- PCA의 주성분 : 위에서 말하는 고유벡터. 입력 데이터의 분산이 가장 큰 방향을 나타낸다.
- 고유값(eigenvalue) : 고유벡터의 크기. 입력 데이터의 분산을 나타냄
- 선형 변환 : 특정 벡터에 행렬 A를 곱해 새로운 벡터로 변환하는 것, 특정 벡터를 하나의 공간(행렬을 공간으로 가정)에서 다른 공간으로 투영하는 개념
- 고유 벡터 : 행렬A를 곱하더라도 방향이 변하지 않고, 그 크기만 변하는 벡터
  - Ax = ax (A: 행렬, x: 고유 벡터, a: 스칼라 값)
  - 이 고유 벡터는 여러 개가 존재하며, 정방 행렬은 최대 그 차원 수 만큼 고유 벡터를 가질 수 있다. (예: 2x2 행렬은 최대 2개의 고유벡터를 가질 수 있음, 3x3은 3개)
  - 이렇듯 고유벡터는 행렬이 작용하는 힘의 방향과 관계가 있어서, **행렬을 분해하는 데 사용**됨
- 분산 : 한 개의 특정한 변수의 데이터 변동을 의미
- 공분산 : 두 변수 간의 변동을 의미
  - 사람의 키 변수를 X, 몸무게 변수를 Y로 둘 때, 공분산 $Cov(X,Y)>0$ == X(키)가 증가할 때 Y(몸무게)도 증가한다는 의미
- 공분산 행렬 : 여러 변수와 관련된 공분산을 포함하는, 정방 행렬 & 대칭 행렬

	X	Y	Z
X	**3.0**	-0.71	-0.24
Y	-0.71	**4.5**	0.28
Z	-0.24	0.28	**0.91**

  - **대각선 원소**는 각 변수(X, Y, Z)의 분산을 의미
  - 대각선 외의 원소는, 가능한 모든 변수 쌍 간의 공분산을 의미
  - X와 Y의 공분산 = -0.71
- 정방 행렬(Diagonal Matrix) : 열과 행이 같은 행렬
- 대칭 행렬(Symmetric Matrix) : 정방 행렬 중에서 대각 원소를 중심으로 원소값이 대칭되는 행렬, $AT=A$대
  - 대칭 행렬은 항상 고유 벡터를 직교 행렬로, 고유값을 정방 행렬로 대각화 할 수 있음 ⇒ 고유값 분해
- 공분산 행렬의 분해
  - $C=P\sum PT$

$C=[e1 \cdots en]\lfloor\lfloor\lceil\lambda 1 \cdots 0 \cdots\cdots\cdots 0 \cdots \lambda n\rfloor\rfloor\rceil\lfloor\lfloor\lceil e1t \cdots ent\rfloor\rfloor\rceil$

C = 고유벡터의 직교 행렬 *고유값 정방행렬* 고유벡터 직교행렬의 전치 행렬

- ei 는 i 번째 고유 벡터
- λi 는 i 번째 고유벡터의 크기(고유값)
- $e1$ 는 가장 분산이 큰 방향을 가진 고유 벡터
- $e2$ 는 $e1$ 에 수직이면서, 그 다음으로 분산이 큰 방향을 가진 고유벡터

- **PCA : 입력 데이터의 공분산 행렬이 고유벡터와 고유값으로 분해될 수 있으며, 이렇게 분해된 고유벡터를 이용해 입력 데이터를 선형 변환하는 방식**
- 수행
  1. 입력 데이터 세트의 공분산 행렬 생성 (C)
  2. 공분산 행렬의 고유벡터(ei)와 고유값(λi)을 계산
  3. 고유값(λi)이 가장 큰 순으로 K개(PCA 변환 차수)만큼 고유벡터(ei)를 추출
  4. 고유값(λi)이 가장 큰 순으로 추출된 고유벡터(ei)를 이용해 새롭게 입력 데이터 변환

🩴 PCA의 구성 개념 정리

- 입력 데이터의 공분산 행렬을 고유값 분해하고, 이렇게 구한 고유벡터(주성분)에 입력 데이터를 선형 변환하는 것
  - 주성분 : 위에서 말하는 고유벡터. 입력 데이터의 분산이 가장 큰 방향을 나타낸다.
  - 공분산 (행렬) : 두 변수 간의 변동을 의미 (여러 변수와 관련된 공분산을 포함하는 대칭 행렬)
  - 공분산 행렬은 항상 고유 벡터를 직교 행렬로, 고유값을 정방 행렬로 대각화 할 수 있음 ⇒ 고유값 분해
  - 고유값 (λ) : 고유벡터의 크기, 입력 데이터의 분산을 나타냄
  - 고유 벡터 (e) : 행렬A 곱해도, 방향 변화 X & 크기만 변화 O 벡터, 행렬이 작용하는 힘의 방향과 관계 있음
  - 선형 변환 : 특정 벡터(고유 벡터)에 행렬 A(입력 데이터)를 곱해 새로운 벡터로 변환하는 것
    - 특정 벡터를 하나의 공간(행렬을 공간으로 가정)에서 다른 공간으로 투영하는 개념

- 공분산 행렬의 분해
  - $C = [e_1 \cdots e_n] \begin{bmatrix} \lambda_1 & \cdots & 0 \\ \cdots & \cdots & \cdots \\ 0 & \cdots & \lambda_n \end{bmatrix} \begin{bmatrix} e_1^t \\ \cdots \\ e_n^t \end{bmatrix}$

    $[e_1 \cdots e_n]$ = 고유 벡터 : 행렬이 작용하는 힘의 방향

    $\begin{bmatrix} \lambda_1 & \cdots & 0 \\ \cdots & \cdots & \cdots \\ 0 & \cdots & \lambda_n \end{bmatrix}$ = 고유값 : 입력데이터의 분산 & 고유 벡터의 크기

  - e_1 는 가장 분산이 큰 방향을 가진 고유 벡터, e_2 는 e_1 에 수직이면서, 그 다음으로 분산이 큰 방향을 가진 고유벡터

- PCA를 적용하기 위해서는 각 속성값을 동일한 스케일로 변환해야 한다. ⇒ StandardScaler
  - 여러 속성 값을 연상해야 하므로, 속성의 스케일에 영향을 받기 때문

```
from sklearn.preprocessing import StandardScaler

# Target 값을 제외한 모든 속성 값을 StandardScaler를 이용하여 표준 정규 분포를 가지는 값들로 변환
iris_scaled = StandardScaler().fit_transform(irisDF.iloc[:, :-1])

from sklearn.decomposition import PCA

pca = PCA(n_components=2)

#fit( )과 transform( ) 을 호출하여 PCA 변환 데이터 반환
pca.fit(iris_scaled)
iris_pca = pca.transform(iris_scaled)
print(iris_pca.shape)

# PCA 환된 데이터의 컬럼명을 각각 pca_component_1, pca_component_2로 명명
pca_columns=['pca_component_1','pca_component_2']
irisDF_pca = pd.DataFrame(iris_pca, columns=pca_columns)
irisDF_pca['target']=iris.target
irisDF_pca.head(3)
```

  - n_components: PCA로 변환할 차원의 수
  - 이후 fit()과 transform()을 호출해 PCA 변환 데이터 반환
  - explained*variance_ratio*: 전체 변동성에서 개별 PCA 컴포넌트별로 차지하는 변동성 비율 제공

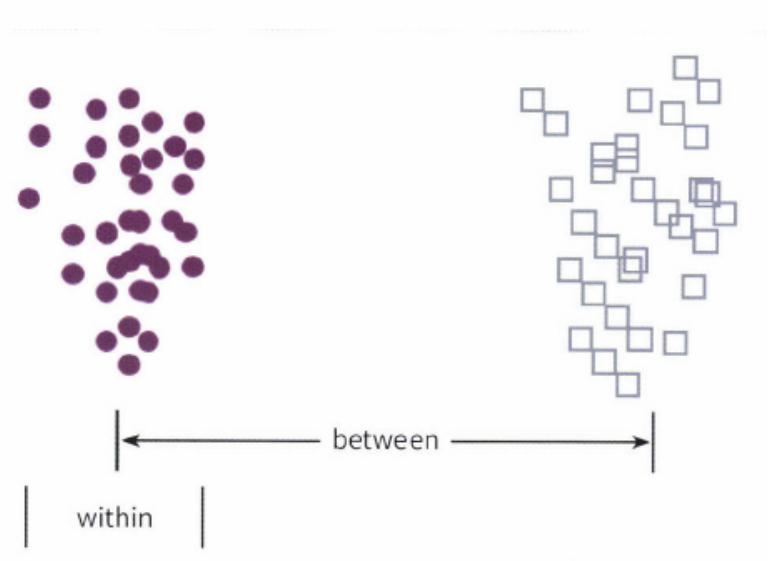
2️⃣ LDA(Linear Discriminant Analysis)

2-1. LDA 개요

PCA와 유사하지만, 지도 학습의 분류에서 사용하기 쉽도록 개별 클래스를 분별할 수 있는 기준을 최대한 유지하면서 차원 축소

PCA	LDA
입력 데이터 세트를 저차원 공간에 투영해 차원을 축소하는 기법	
비지도학습	지도학습의 분류에서 사용하기 쉽도록 개별 클래스를 분별할 수 있는 기준을 최대한 유지하면서 차원 축소
입력 데이터의 변동성이 가장 큰 축	입력 데이터의 결정 값 클래스를 최대한으로 분리할 수 있는 축 ->클래스 간 분산(between-class scatter)과 클래스 내부 분산(within-class scatter)의 비율을 최대화하는 방식으로 차원을 축소

- 입력 데이터의 결정값 클래스를 최대한으로 분리할 수 있는 축을 찾기 위해, 클래스 간 분산(between)과 클래스 내부 분산(within)의 비율을 최대화하는 방식으로 차원 축소



클래스 간 분산은 크게, 클래스 내부 분산은 작게

[LDA step]

1. 클래스 내부와 클래스 간 분산 행렬을 구한다. 이 두 개의 행렬은 입력 데이터의 결정 값 클래스별로 개별 피처의 평균 벡터(mean vector)를 기반으로 구한다.
2. 클래스 내부 분산 행렬을 Sw, 클래스 간 분산 행렬을 Sb라고 하면 두 행렬을 고유벡터로 분해할 수 있다.
3. 고유값이 가장 큰 순으로 k개(LDA변환 차수만큼) 추출한다.
4. 고유값이 가장 큰 순으로 추출된 고유벡터를 이용해 새롭게 입력 데이터를 반환한다.

$$S_W^T S_B = \begin{bmatrix} e_1 & \cdots & e_n \end{bmatrix} \begin{bmatrix} \lambda_1 & \cdots & 0 \\ \cdots & \cdots & \cdots \\ 0 & \cdots & \lambda_n \end{bmatrix} \begin{bmatrix} e_1^T \\ \cdots \\ e_n^T \end{bmatrix}$$

- 사용

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_iris

iris = load_iris()
iris_scaled = StandardScaler().fit_transform(iris.data)

lda = LinearDiscriminantAnalysis(n_components=2)
lda.fit(iris_scaled, iris.target) # 지도학습이라, fit할 때, 클래스 결정값 (y) 넣어야함
iris_lda = lda.transform(iris_scaled)
print(iris_lda.shape)
```

3️⃣ SVD(Singular Value Decomposition, 특이값 분해)

3-1. SVD 개요

PCA의 경우 정방행렬(행 크기 = 열 크기)만을 고유벡터로 분해할 수 있지만, SVD는 정방행렬뿐만 아니라 행과 열의 크기가 다른 행렬에도 적용할 수 있다.

- Full SVD
  - $A=U\sum VT$

- *A* : 행렬, *U*, *V*: 특이벡터(Singular vector)로 된 행렬, ∑ : 대각행렬

  ** 모든 특이벡터는 서로 직교하는 성질

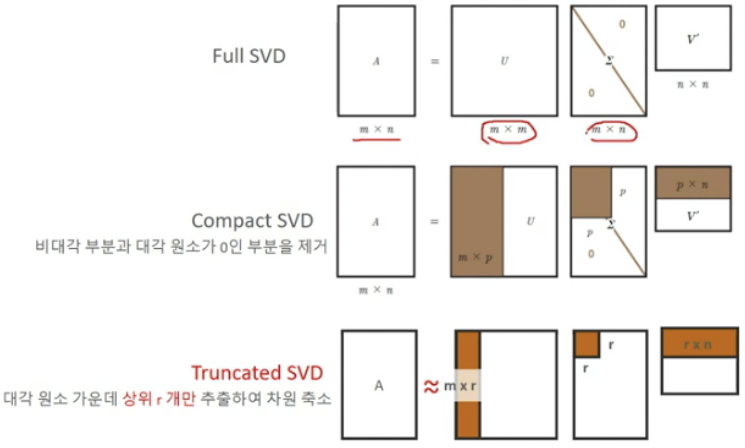
- ∑ : 대각행렬 : 행렬의 대각에 위치한 값만 0이 아니고 나머지 위치의 값은 모두 0인 행렬. 여기서 0이 아닌 값이 행렬 A의 특이값

$$\mathbf{AB} = \mathbf{BA} = \begin{bmatrix} a_1 b_1 & 0 & \cdots & 0 \\ 0 & a_2 b_2 & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \cdots & a_n b_n \end{bmatrix}$$

(A,B는 각각 대각행렬)

- *A* : MxN 행렬일 때 —분해→ *U*: MxM 행렬, ∑ : MxN 행렬, *VT*: NxN 행렬

- Compact SVD (일반적)
  - *U* : MxP 행렬, ∑ : PxP 행렬, *VT*: PxN 행렬
  - ∑ 의 비대각인 부분과 대각원소 중에 특이값이 0인 부분도 모두 제거되고,제거된 ∑에 대응되는 *U*와 *V*원소도 함께 제거해 차원을 줄인 형태로 SVD를 적용한다.
- Truncated SVD
  - 특이값 중 상위 일부 데이터만 추출해 분해하는 방식.
  - 인위적으로 더 작은 차원의 행렬들로 분해하기 때문에 원본행렬을 정확하게는 복원할 수 없다.



- SVD 사용: 보통 넘파이나 사이파이 라이브러리를 이용

```
from numpy.linalg import svd
# or
from scipy.linalg import svd

# numpy의 svd 모듈 import
import numpy as np
from numpy.linalg import svd

# 4X4 Random 행렬 a 생성
np.random.seed(121)
a = np.random.randn(4,4)
print(np.round(a, 3))

> [[-0.212 -0.285 -0.574 -0.44 ]
> [-0.33   1.184  1.615  0.367]
> [-0.014  0.63   1.71  -1.327]
> [ 0.402 -0.191  1.404 -1.969]]

U, Sigma, Vt = svd(a) # a = 원본 행렬
print(U.shape, Sigma.shape, Vt.shape)
print('U matrix:\n',np.round(U, 3))
print('Sigma Value:\n',np.round(Sigma, 3)) # 대각행렬 => 대각에 위치한 값 == 1,  외에 나머지 값 0
print('V transpose matrix:\n',np.round(Vt, 3))

> (4, 4) (4,) (4, 4)
> U matrix:
>  [[-0.079 -0.318  0.867  0.376]
>  [ 0.383  0.787  0.12   0.469]
>  [ 0.656  0.022  0.357 -0.664]
>  [ 0.645 -0.529 -0.328  0.444]]

> Sigma Value:
>  [3.423 2.023 0.463 0.079]

> V transpose matrix:
>  [[ 0.041  0.224  0.786 -0.574]
>  [-0.2    0.562  0.37   0.712]
>  [-0.778  0.395 -0.333 -0.357]
>  [-0.593 -0.692  0.366  0.189]]
```

$$U * \sum * V^T = A$$

- 위처럼 다시 행렬 A로 복원해보기

```
# Sima를 다시 0 을 포함한 대칭행렬로 변환
Sigma_mat = np.diag(Sigma)
a_ = np.dot(np.dot(U, Sigma_mat), Vt)
print(np.round(a_, 3))
```

- Truncated SVD : ∑ 행렬에 있는 대각원소, 즉 특이값 중 상위 일부 데이터만 추출해 분해하는 방식
    - 이렇게 분해하면, 인위적으로 더 작은 차원의 U,∑,VT 로 분해하기에, 원본 행렬을 정확히 원복할 수는 없음
    - 그러나, 데이터 정보가 압축되어 분해됨에도 불구하고 상당한 수준으로 워본 행렬을 근사할 수 있음
- Truncated SVD 사용 : 사이파이에서만 지원됨

```
from scipy.sparse.linalg import svds
```

    - 검증 수행 순서

      1. 임의의 원본 행렬 6x6을 Normal SVD로 분해해 ⇒ 행렬의 차원, ∑ 행렬 내 특이값 확인

      2. 다시 Truncated SVD로 분해해 ⇒ 행렬의 차원, ∑ 행렬 내 특이값 확인

      3. Truncated SVD로 분해된 행렬의 내적을 계산해서 ⇒ 원상 복구하여 원본데이터와 비교

사이킷런 TruncatedSVD 클래스를 이용한 변환

- 사이파이의 SVDs와 같이 U,∑,VT 행렬을 반환하지는 않음.
- 사이킷런의 PCA 클래스와 유사하게, `fit()`, `transform()`으로 원본 데이터를 몇 개의 주요 컴포넌트로 차원축소해 변환
- 즉, 원본 데이터를 Truncated SVD 방식으로 분해된 U*∑ 행렬에 선형변환하여 생성

```
from sklearn.decomposition import TruncatedSVD, PCA
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
%matplotlib inline

iris = load_iris()
iris_ftrs = iris.data
# 2개의 주요 component로 TruncatedSVD 변환
tsvd = TruncatedSVD(n_components=2)
tsvd.fit(iris_ftrs)
iris_tsvd = tsvd.transform(iris_ftrs)

# 2개의 주요 component로 TruncatedSVD 변환 (비교를 위해)
pca = PCA(n_components=2)
pca.fit(iris_ftrs)
iris_pca = pca.transform(iris_ftrs)
```

```
# TruncatedSVD 변환 데이터를 왼쪽에, PCA변환 데이터를 오른쪽에 표현
fig, (ax1, ax2) = plt.subplots(figsize=(18,4), ncols=2)
ax1.scatter(x=iris_tsvd[:,0], y= iris_tsvd[:,1], c= iris.target)
ax2.scatter(x=iris_pca[:,0], y= iris_pca[:,1], c= iris.target)
ax1.set_title('Truncated SVD Transformed')
ax2.set_title('PCA Transformed')
ax1.set_xlabel('TruncatedSVD Component 1')
ax1.set_ylabel('TruncatedSVD Component 2')
ax2.set_xlabel('PCA Component 1')
ax2.set_ylabel('PCA Component 2')
```

```
print((iris_pca - iris_tsvd).mean())
print((pca.components_ - tsvd.components_).mean())

> 2.3419865583888347e-15
> 6.245004513516506e-17
```

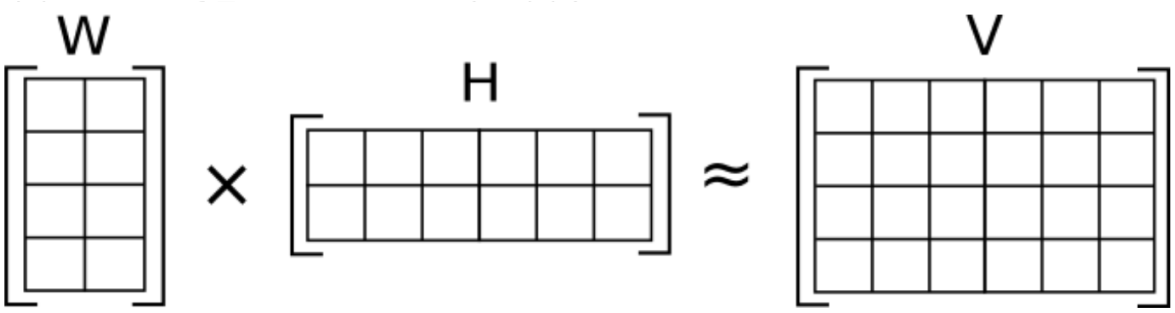
- 모두 0에 가까운 값이므로, 2개의 변환이 서로 동일함을 알 수 있음
- 즉, 데이터 세트가 스케일링으로 데이터 중심이 동일해지면, SVD와 PCA는 동일한 변환을 수행
- 이는 PCA가 SVD 알고리즘으로 구현됐음을 의미
- 그러나, PCA는 밀집 행렬(Dense Matrix)에 대한 변환만 가능하며, SVD는 희소 행렬(Sparse Matrix)에 대한 변환도 가능
- 또한 SVD는 텍스트의 토픽 모델링 기법인 LSA(Latent Semantic Analysis)의 기반 알고리즘임.

4️⃣ NMF(N0n-Negative Matrix Factorization)

4-1. NMF 개요

원본 행렬 내의 모든 원소값이 모두 양수(0 이상)라는 게 보장되면, 두 개의 기반 양수 행렬로 분해될 수 있는 기법

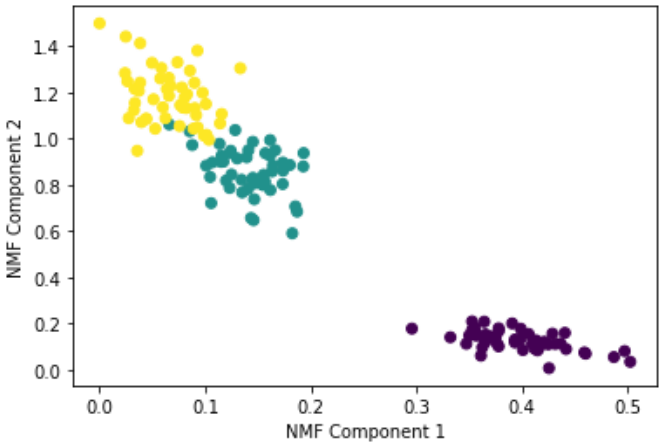
➡️Truncated SVD와 같이 낮은 랭크를 통한 행렬 근사 방식의 변형



- $W \times H \approx V$
  - 일반적으로 길고 가는 행렬 W(원본 행렬과 행크기 같고 열크기 보다 작은 행렬) X 작고 넓은 행렬 H (원본 행렬의 행 크기보다 작고 열 크기와 같은 행렬)로 분해된다.
  - W : 원본 행에 대해서 이 잠재요소의 값이 얼마나 되는지에 대응
  - H : 이 잠재요소가 원본 열(원본 속성)로 어떻게 구성됐는지를 나타냄
- 사용

```
from sklearn.decomposition import NMF
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
%matplotlib inline

iris = load_iris()
iris_ftrs = iris.data
nmf = NMF(n_components=2)
nmf.fit(iris_ftrs)
iris_nmf = nmf.transform(iris_ftrs)
plt.scatter(x=iris_nmf[:,0], y= iris_nmf[:,1], c= iris.target)
plt.xlabel('NMF Component 1')
plt.ylabel('NMF Component 2')
```



- NMF와 SVD와 유사하게 이미지 압축을통한 패턴 인식, 텍스트의 토픽 모델링 기법, 문서 유사도 및 클러스터링, 추천 시스템에 활발히 적용 됨

5️⃣ 정리

- PCA
  - 입력 데이터의 변동성이 가장 큰 축을 구하고, 다시 이 축에 직각인 축을 반복적으로 축소하려는 차원의 개수만큼 구한 뒤 입력 데이터를 이 축들에 투영해 차원을 축소하는 방식
  - 입력 데이터의 공분산 행렬을 기반으로, 고유 벡터를 생성하고, 이 고유 벡터에 입력 데이터를 선형변환하는 방식
- LDA
  - 입력 데이터의 결정값 클래스를 최대한으로 분리할 수 있는 축을 찾아 차원을 축소하는 방식
- SVD, NMF
  - 고차원 행렬을 두 개의 저차원 행렬로 분리하는 행렬기법
  - 원본 행렬에서 잠재된 요소를 추출하기 때문에 토픽 모델이나 추천시스템에서 사용됨