차원축소

– 권수태 교수

JEONJU UNIVERSITY
The Place for Superstars

1. 차원축소

❖차원축소 개요

- 매우 많은 피처로 구성된 다차원 데이터 세트의 차원을 축소해 새로운 차원의 데이터 세트를 생성하는 것
  - ✓ 일반적으로 차원이 증가하면 데이터 간의 거리가 멀어지고(차원의 저주) , 희소한 구조를 갖게 됨
  - ✓ 적은 차원에서 학습된 모델보다 상대적으로 예측 신뢰도가 떨어짐
  - ✓ 피처가 많은 경우 개별 피처간에 상관관계가 높을 가능성 존재 => 다중 공선성으로 인한 예측 성능 저하 우려

    => 차원 축소하여 피처의 수를 줄이면 더 직관적으로 데이터 해석 가능
- 수십개 이상의 피처는 시각적으로 표현해 특성을 파악하기 힘드므로 3차원 이하 차원 축소를 통해 시각적으로 데이터 압축하여 표현

=> 쉽게 데이터 패턴 인지 가능, 학습 데이터 크기가 줄어들어 학습에 필요한 처리 능력 줄일 수 있음

1. 차원축소

❖차원축소 개요

➢ 차원축소는 피처 선택과 피처 추출로 나누어짐

✓ 피처 선택 : 특정 피처에 종속성이 강한 불필요한 피처는 아예 제거하고, 데이터의 특징을 잘 나타내는 주요 피처만 선택하는 것

✓ 피처(특성) 추출 : 기존 피처를 저차원의 중요 피처로 압축해서 추출

- 기존 피처가 압축된 것이므로 기존의 피처와는 완전히 다른 값이 되며, 피처를 함축적으로 더 잘 설명할 수 있는 또 다른 공간으로 매핑해 추출함
- 기존 피처가 전혀 인지하기 어려웠던 잠재적인 요소(Latent Factor)를 추출하는 것
- 이미지, 텍스트처럼 많은 차원을 갖고 있는 곳에서 잘 활용

1. 차원축소

❖차원축소 개요

➢ 차원축소는 피처 선택과 피처 추출로 나누어짐

✓ 피처(특성) 추출

- 이미지, 텍스트처럼 많은 차원을 갖고 있는 곳에서 잘 활용
- 이미지 데이터에서 잠재된 특성을 피처로 도출해 함축적 형태의 이미지 변환과 압축 수행 가능
- 변환된 이미지는 원본보다 훨씬 적은 차원이기 때문에 과적합 영향력이 작아져서 원본으로 예측하는 것보다 예측성능 향상 가능
- 문서내 단어들의 구성에서 숨겨져 있는 시맨틱 의미나 토픽을 잠재요소로 간주하고 이를 찾아 낼 수 있음
- SVD 나 NMF 는 시맨틱 토픽모델링을 위한 기반 알고리즘으로 사용됨

1. 차원축소

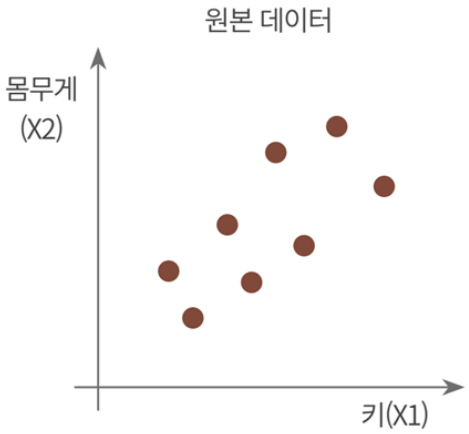
❖PCA(Principal Component Analysis, 주성분 분석)

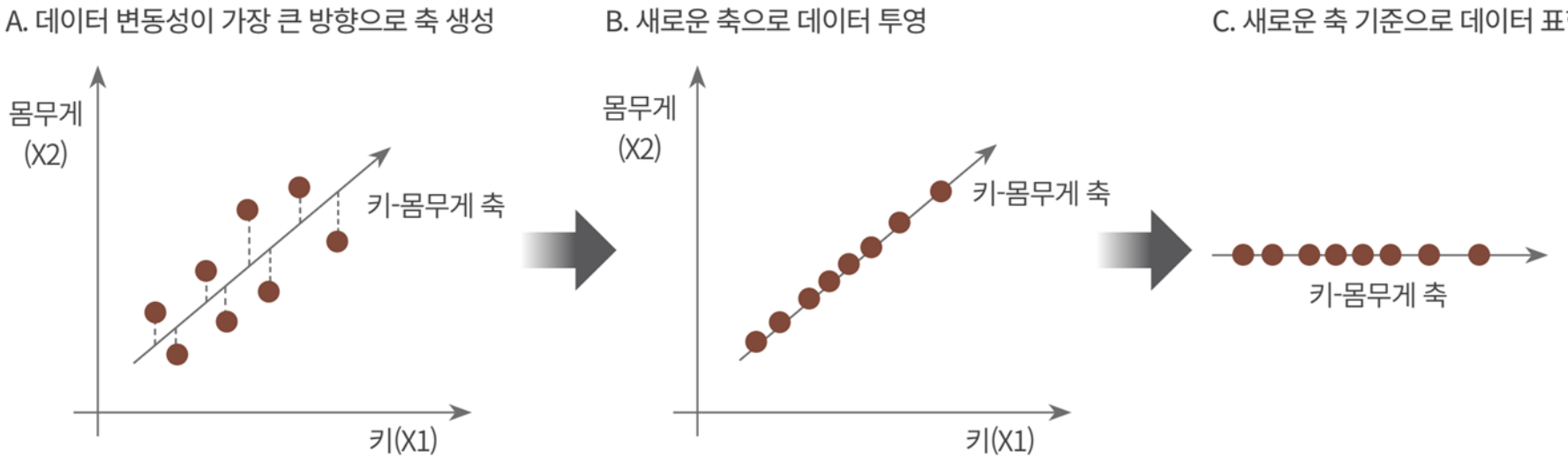
- ➢ 가장 대표적인 차원축소 기법
- ➢ 여러 변수 간에 존재하는 상관관계를 이용해 이를 대표하는 주성분을 추출해 차원을 축소하는 기법
- ➢ 고차원의 원본 데이터를 저차원의 부분 공간으로 투영하여 데이터 축소
- ➢ PCA로 차원 축소 시 기존 데이터의 정보 유실이 최소화됨
- ➢ 가장 높은 분산을 가지는 데이터의 축을 찾아 이 축으로 차원을 축소

=> PCA의 주성분 (∴분산이 데이터의 특성을 가장 잘 나타내는 것으로 간주)

1. 차원축소

❖PCA(Principal Component Analysis, 주성분 분석)





JEONJU UNIVERSITY

1. 차원축소

❖PCA(Principal Component Analysis, 주성분 분석)

- ➢차원 축소 방법 : 데이터 변동성이 가장 큰 방향으로 축 생성 -> 두번째 축은 이 벡터 축에 직각이 되는 벡터(직교 벡터)를 축으로 -> 세번째 축은 다시 두번째 축과 직각이 되는 벡터 ->... ->새롭게 생성된 축으로 데이터 투영
- ➢PCA는 매우 작은 주성분으로 원본 데이터의 총 변동성을 대부분 설명할 수 있는 분석법
- ➢선형대수 관점에서 해석해보면, 입력 데이터의 공분산 행렬을 고유값 분해하고 이렇게 구한 고유 벡터에 입력 데이터를 선형변환하는 것
- ➢여기서 이 고유 벡터가 PCA의 주성분 벡터로서 입력 데이터의 분산이 큰 방향을 나타냄

1. 차원축소

❖PCA(Principal Component Analysis, 주성분 분석)

- ➢ 선형 변환
  - ✓ 특정 벡터에 행렬 A를 곱해 새로운 벡터로 변환하는 것
  - ✓ 특정 벡터를 하나의 공간에서 다른 공간으로 투영하는 개념(이 경우 행렬은 공간)
- ➢ 고유벡터
  - ✓ 행렬을 곱해도 방향은 변하지 않고 크기만 변하는 벡터 (Ax = ax)
  - ✓ 고유벡터는 여러개가 존재하며 , 정방 행렬은 최대 차원수만큼 고유벡터 가질 수 있음
- ➢ 공분산 행렬
  - ✓ 정방행렬이며 대칭행렬 / 개별 분산값을 대각 원소로 하는 대칭행렬
- ➢ 대칭행렬
  - ✓ 항상 고유벡터를 직교행렬로, 고유값을 정방 행렬로 대각화할 수 있음

1. 차원축소

❖PCA(Principal Component Analysis, 주성분 분석)

- ➢공분산 C는 고유벡터 직교행렬 * 고유값 정방 행렬 * 고유벡터 직교행렬의 전치행렬로 분해됨
  - ✓고유벡터 e에서 e1은 가장 분산이 큰 방향을 가진 고유벡터
  - ✓고유값 λi 는 i번째 고유벡터의 크기
- ➢입력 데이터의 공분산 행렬이 고유 벡터와 고유값으로 분해되며 이렇게 분해된 고유벡터를 이용해 입력 데이터를 선형변환하는 것이 PCA

$$C = P \sum P^T \qquad C = [e_1 \cdots e_n] \begin{bmatrix} \lambda_1 & \cdots & 0 \\ \cdots & \cdots & \cdots \\ 0 & \cdots & \lambda_n \end{bmatrix} \begin{bmatrix} e_1^t \\ \cdots \\ e_n^t \end{bmatrix}$$

1. 차원축소

❖PCA(Principal Component Analysis, 주성분 분석)

- PCA과정
  - 1) 입력 데이터 세트의 공분산 행렬을 생성
  - 2) 공분산 행렬의 고유벡터와 고유값 계산
  - 3) 고유값이 가장 큰 순으로 K개 만큼 고유벡터 추출
  - 4) 고유값이 가장 큰 순으로 추출된 고유벡터를 이용해 새롭게 입력 데이터 변환
- PCA를 적용하기 전 개별 속성을 함께 스케일링 해야함
  - ✓ PCA는 여러 속성의 값을 연산하므로 속성의 스케일에 영향 받음
  - ✓ 여러 속성을 PCA로 압축하기 전에 각 속성값을 동일한 스케일로 변환하는 것 필요
- 사이킷런은 PCA 변환을 위해 PCA 클래스 제공
  - ✓ explained_variance_ratio_: 전체 변동성에서 개별로 차지하는 변동성 비율

1. 차원축소

❖PCA(Principal Component Analysis, 주성분 분석)

➢ 붓꽃 데이터를 이용해 4개의 속성을 2개의 PCA 차원으로 압축해 원래 데이터 세트와 압축된 데이터 세트가 어떻게 달라졌는지 확인

```
from sklearn.datasets import load_iris
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

# 사이킷런 내장 데이터 셋 API 호출
iris = load_iris()

# 넘파이 데이터 셋을 Pandas DataFrame으로 변환
columns = ['sepal_length','sepal_width','petal_length','petal_width']
irisDF = pd.DataFrame(iris.data , columns=columns)
irisDF['target']=iris.target
irisDF.head(3)
```


1. 차원축소

❖PCA(Principal Component Analysis, 주성분 분석)

- 각 품종에 따라 원본 붓꽃 데이터 세트가 어떻게 분포되어 있는지 2차원으로 시각화

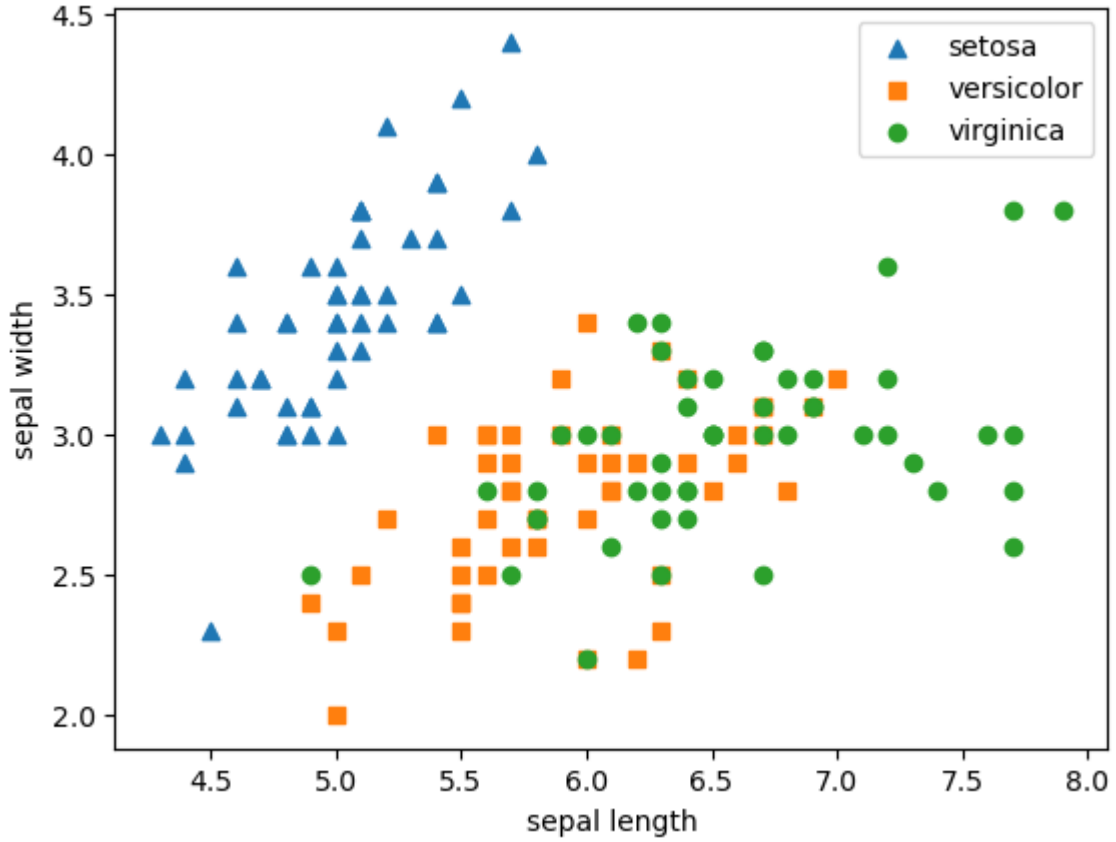
- sepal length와 sepal width를 X,Y 축으로 해 품종 데이터 분포를 나타냄

```
#setosa는 세모, versicolor는 네모, virginica는 동그라미로 표현
markers=['^', 's', 'o']

#setosa의 target 값은 0, versicolor는 1, virginica는 2. 각 target 별로 다른 shape으로 scatter plot
for i, marker in enumerate(markers):
    x_axis_data = irisDF[irisDF['target']==i]['sepal_length']
    y_axis_data = irisDF[irisDF['target']==i]['sepal_width']
    plt.scatter(x_axis_data, y_axis_data, marker=marker,label=iris.target_names[i])

plt.legend()
plt.xlabel('sepal length')
plt.ylabel('sepal width')
plt.show()
```


1. 차원축소

❖PCA(Principal Component Analysis, 주성분 분석)

➢ sepal length와 sepal width를 X,Y 축으로 해 품종 데이터 분포를 나타냄



JEONJU UNIVERSITY

1. 차원축소

❖PCA(Principal Component Analysis, 주성분 분석)

- 여러 속성을 PCA로 압축하기 전에 각 속성값을 동일한 스케일로 변환하는 것 필요(PCA는 여러 속성의 값을 연산하므로 속성의 스케일에 영향 받음)
- StandardScaler를 이용해 평균이 0, 분산이 1인 표준 정규 분포로 속성값 변환

```
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])
```

- PCA로 4개의 속성을 2개로 압축

```
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)
```


1. 차원축소

❖PCA(Principal Component Analysis, 주성분 분석)

- Iris_pca 는 변환된 PCA 데이터세트를 150*2 넘파이 행렬로 가지고있음
- 이를 DataFrame 으로 변환한 뒤 데이터값을 확인

```
# 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)
```

	pca_component_1	pca_component_2	target
0	-2.264703	0.480027	0
1	-2.080961	-0.674134	0
2	-2.364229	-0.341908	0

1. 차원축소

❖PCA(Principal Component Analysis, 주성분 분석)

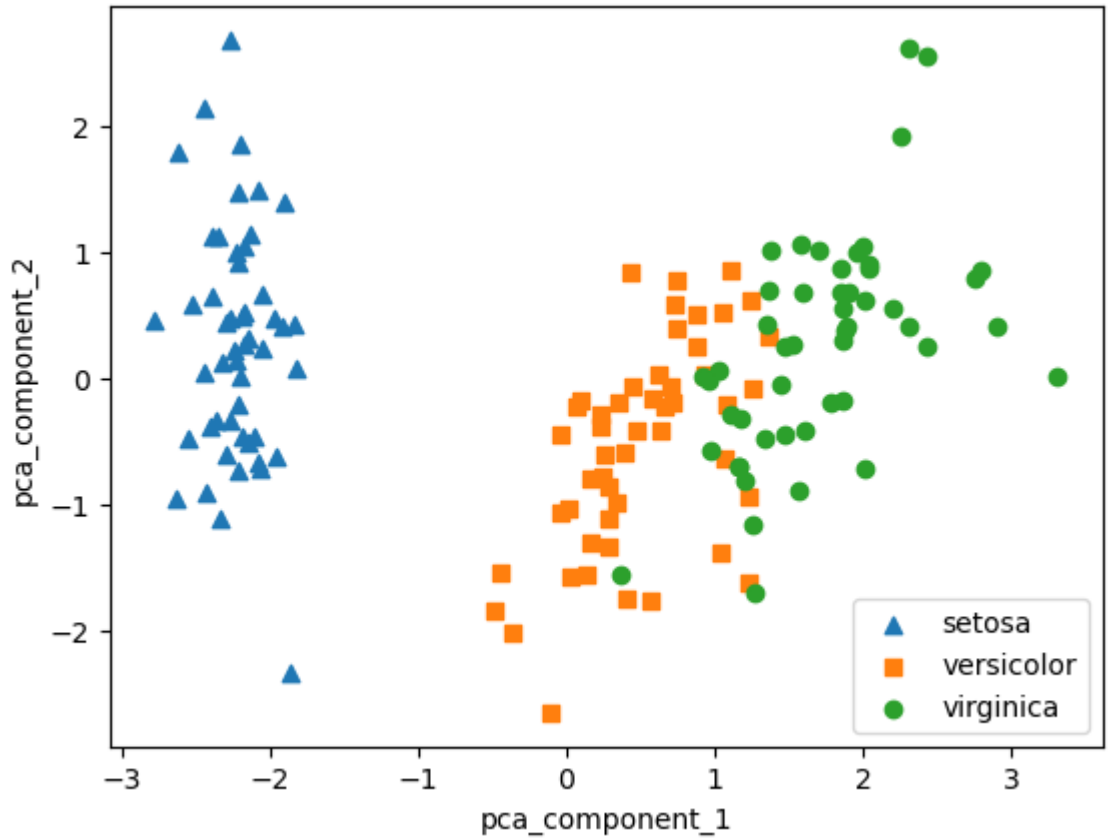
➢ 2개의 속성으로 PCA 변환된 데이터 세트를 2차원상에서 시각화

```
#setosa를 세모, versicolor를 네모, virginica를 동그라미로 표시
markers=['^', 's', 'o']

#pca_component_1 을 x축, pc_component_2를 y축으로 scatter plot 수행.
for i, marker in enumerate(markers):
    x_axis_data = irisDF_pca[irisDF_pca['target']==i]['pca_component_1']
    y_axis_data = irisDF_pca[irisDF_pca['target']==i]['pca_component_2']
    plt.scatter(x_axis_data, y_axis_data, marker=marker,
label=iris.target_names[i])

plt.legend()
plt.xlabel('pca_component_1')
plt.ylabel('pca_component_2')
plt.show()
```


1. 차원축소

❖PCA(Principal Component Analysis, 주성분 분석)

- 2개의 속성으로 PCA 변환된 데이터 세트를 2차원상에서 시각화



1. 차원축소

❖PCA(Principal Component Analysis, 주성분 분석)

➢ PCA Component 별로 원본 데이터의 변동성을 얼마나 반영하고 있는지 알아본다

```
print(pca.explained_variance_ratio_)
```

```
[0.72962445 0.22850762]
```

첫번째 PCA 변환 요소는 전체 변동성의 약 72.9%를 차지하며,
두번째는 약 22.8%를 차지

그러므로 PCA를 2개의 요소로만 변환해도 원본 데이터의
변동성을 95% 설명

1. 차원축소

❖PCA(Principal Component Analysis, 주성분 분석)

➢ 원본 붓꽃 데이터 세트와 PCA로 변환된 데이터 세트에 각각 분류를 적용한 후 결과 비교

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
import numpy as np

rcf = RandomForestClassifier(random_state=156)
scores = cross_val_score(rcf, iris.data, iris.target, scoring='accuracy', cv=3)
print('원본 데이터 교차 검증 개별 정확도:',scores)
print('원본 데이터 평균 정확도:', np.mean(scores))
```

```
pca_X = irisDF_pca[['pca_component_1', 'pca_component_2']]
scores_pca = cross_val_score(rcf, pca_X, iris.target, scoring='accuracy', cv=3 )
print('PCA 변환 데이터 교차 검증 개별 정확도:',scores_pca)
print('PCA 변환 데이터 평균 정확도:', np.mean(scores_pca))
```

원본 데이터 세트 대비 예측 정확도는 PCA 변환 차원 개수에 따라 예측 성능이 떨어질 수 밖에 없다 (10% 하락)
PCA를 이용하면 성능은 떨어지지만 데이터를 명확하게 표현하는 방향성을 제공

1. 차원축소

❖LDA(Linear Discriminant Analysis)

- ➢ 선형 판별 분석법으로 불리며 PCA와 매우 유사
- ➢ 지도학습의 분류에서 사용하기 쉽도록 개별 클래스를 분별할 수 있는 기준을 최대한 유지하면서 차원축소
- ➢ PCA는 입력 데이터의 변동성의 가장 큰 축을 찾는다면, LDA는 입력 데이터의 결정값 클래스를 최대한으로 분리할 수 있는 축을 찾음
  - ✓ LDA는 지도학습 ! => 클래스의 결정값이 변환 시에 필요함
- ➢ LDA는 특정 공간 상에서 클래스 분리를 최대화하는 축을 찾기 위해 클래스 간 분산과 클래스 내부 분산의 비율을 최대화하는 방식으로 차원 축소 => 클래스 간 분산은 최대한 크게, 클래스 내부의 분산은 최대한 작게
- ➢ 공분산 행렬을 사용하는 PCA와 달리 클래스 간 분산과 클래스 내부 분산 행렬을 생성한 뒤, 이 행렬에 기반해 고유벡터를 구하고 입력 데이터를 투영함

1. 차원축소

❖LDA(Linear Discriminant Analysis)

➢LDA 구하는 과정

1) 클래스 내부와 클래스 간 분산 행렬을 구함. 2개의 행렬은 입력 데이터의 결정 값 클래스 별로 개별 피처의 평균 벡터를 기반으로 구함

2) 클래스 내부 분산 행렬을 Sw, 클래스 간 분산 행렬을 Sb 이라 할 때 이는 고유 벡터로 분해가 가능

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

3) 고유값이 가장 큰 순으로 K개 추출

4) 고유값이 가장 큰 순으로 추출된 고유벡터를 이용해 새롭게 입력 데이터 변환

1. 차원축소

❖LDA(Linear Discriminant Analysis)

➢LDA 구하는 과정

```
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)
iris_lda = lda.transform(iris_scaled)
print(iris_lda.shape)
```


1. 차원축소

❖LDA(Linear Discriminant Analysis)

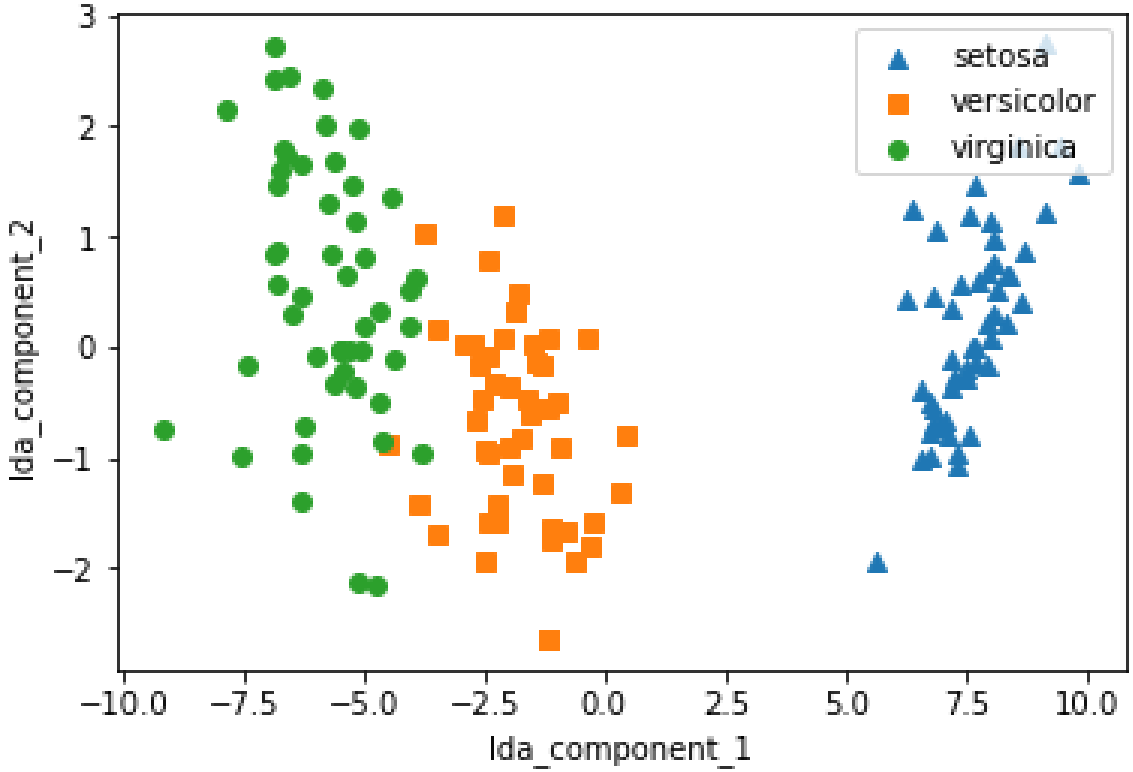
➢LDA 구하는 과정

```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
lda_columns=['lda_component_1','lda_component_2']
irisDF_lda = pd.DataFrame(iris_lda,columns=lda_columns)
irisDF_lda['target']=iris.target
#setosa는 세모, versicolor는 네모, virginica는 동그라미로 표현
markers=['^', 's', 'o']
#setosa의 target 값은 0, versicolor는 1, virginica는 2. 각 target 별로 다른 shape으로 scatter plot
for i, marker in enumerate(markers):
    x_axis_data = irisDF_lda[irisDF_lda['target']==i]['lda_component_1']
    y_axis_data = irisDF_lda[irisDF_lda['target']==i]['lda_component_2']
    plt.scatter(x_axis_data, y_axis_data, marker=marker,label=iris.target_names[i])
plt.legend(loc='upper right')
plt.xlabel('lda_component_1')
plt.ylabel('lda_component_2')
plt.show()
```


1. 차원축소

❖LDA(Linear Discriminant Analysis)

➢LDA 구하는 과정



1. 차원축소

❖SVD(Singular Value Decomposition)

- 정방행렬뿐만 아니라 행과 열의 크기가 다른 행렬에도 행렬 분해 기법 사용가능

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

- 특이값 분해로 불리며, 행렬 U와 V에 속하는 벡터는 특이벡터 , ∑는 대각행렬
- 특이 벡터는 서로 직교하는 성질 가짐
- ∑이 위치한 0이 아닌 값이 행렬의 특이값

1. 차원축소

❖SVD(Singular Value Decomposition)

```
# 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))
```

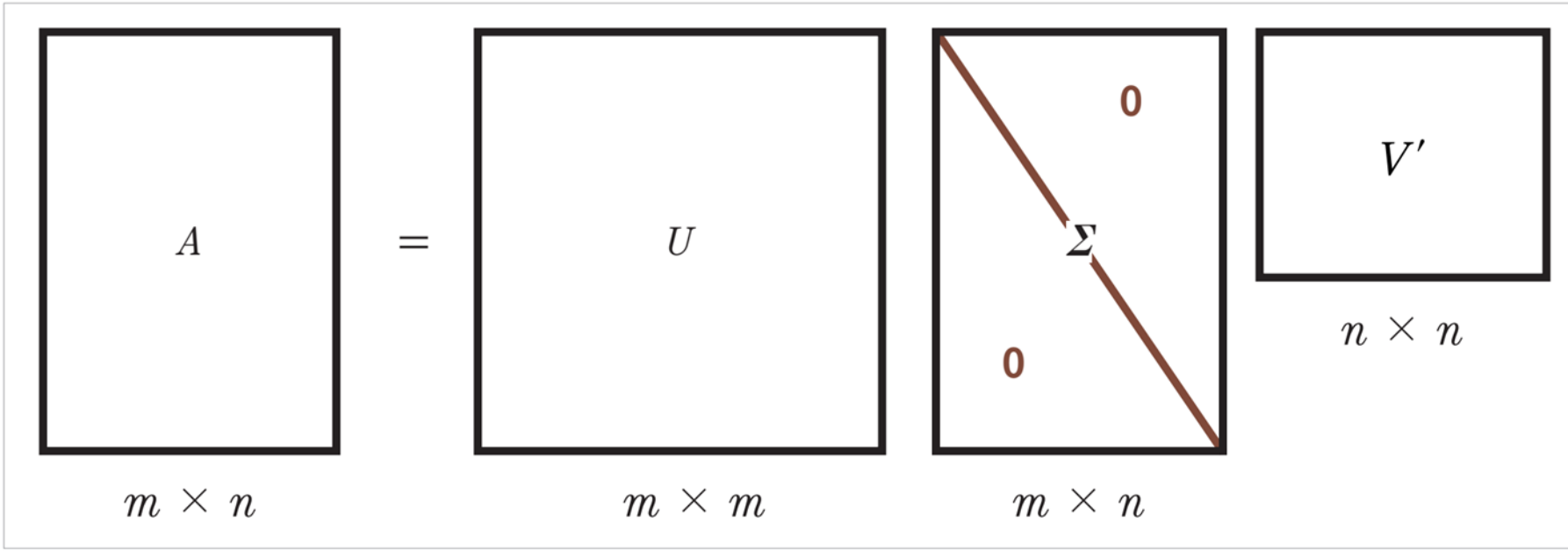
```
U, Sigma, Vt = svd(a)
print(U.shape, Sigma.shape, Vt.shape)
print('U matrix:\n',np.round(U, 3))
print('Sigma Value:\n',np.round(Sigma, 3))
print('V transpose matrix:\n',np.round(Vt, 3))
```

```
# 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))
```


1. 차원축소

❖SVD(Singular Value Decomposition)

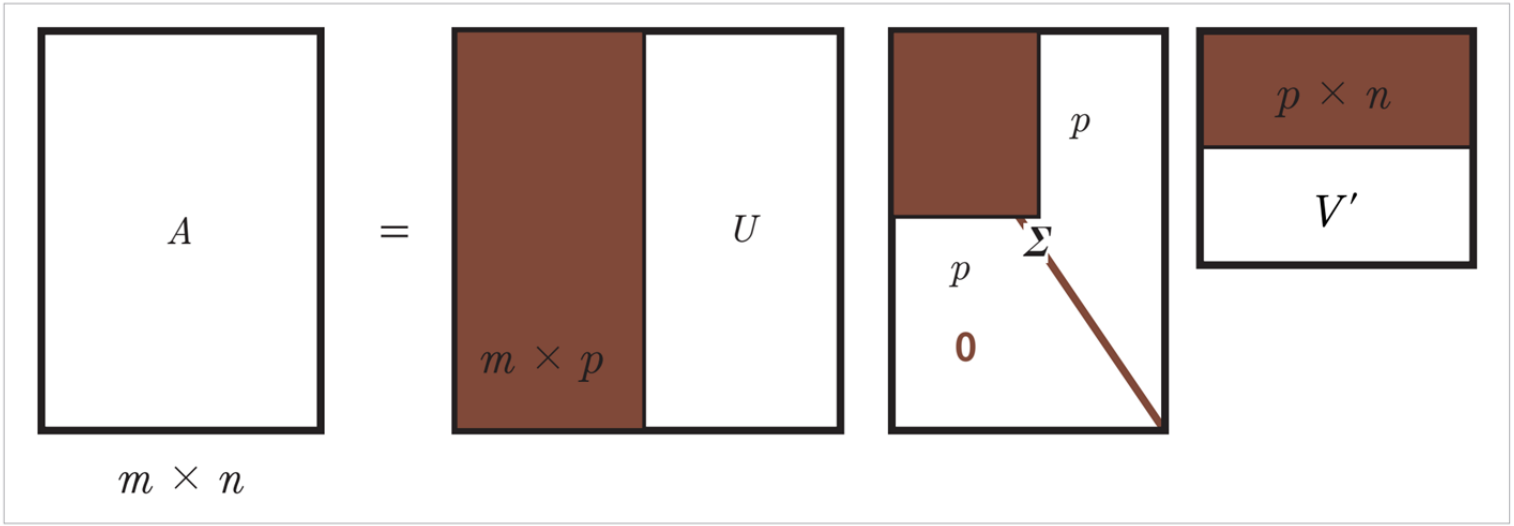
➢ Full SVD



1. 차원축소

❖SVD(Singular Value Decomposition)

- ➢ Compact SVD
  - ✓ ∑의 비대각인 부분과 대각원소 중에 특이값이 0인 부분도 모두 제거하고 이에 해당하는 U와 V의 원소도 함께 제거해 차원을 줄인 형태
- ➢ Truncated SVD : ∑의 대각원소 중에 상위 몇개만 추출



JEONJU UNIVERSITY

1. 차원축소

❖SVD(Singular Value Decomposition)

- ➢ 일반적인 SVD 는 보통 넘파이나 사이파이 라이브러리를 이용해 수행
- ➢ 넘파이의 SVD 이용

```
# 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))
```

```
U, Sigma, Vt = svd(a)
print(U.shape, Sigma.shape, Vt.shape)
print('U matrix:\n',np.round(U, 3))
print('Sigma Value:\n',np.round(Sigma, 3))
print('V transpose matrix:\n',np.round(Vt, 3))
```

```
# 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))
```


1. 차원축소

❖SVD(Singular Value Decomposition)

- ➢ 사이킷런 TruncatedSVD 클래스는 PCA 클래스와 유사하게 fit()와 transform() 를 호출해 원본데이터를 몇 개의 주요 컴포넌트로 차원을 축소해 반환
  - ✓ 사이킷런 TruncatedSVD 클래스는 사이파이의 svds 와 같이 Truncated SVD 연산을 수행해 원본 행렬을 분해한 U, Sigma, Vt 행렬을 반환하지는 않음
- ➢ 원본 데이터를 Truncated SVD 방식으로 분해된 U*Sigma 행렬에 선형 변환해 생성

1. 차원축소

❖SVD(Singular Value Decomposition)

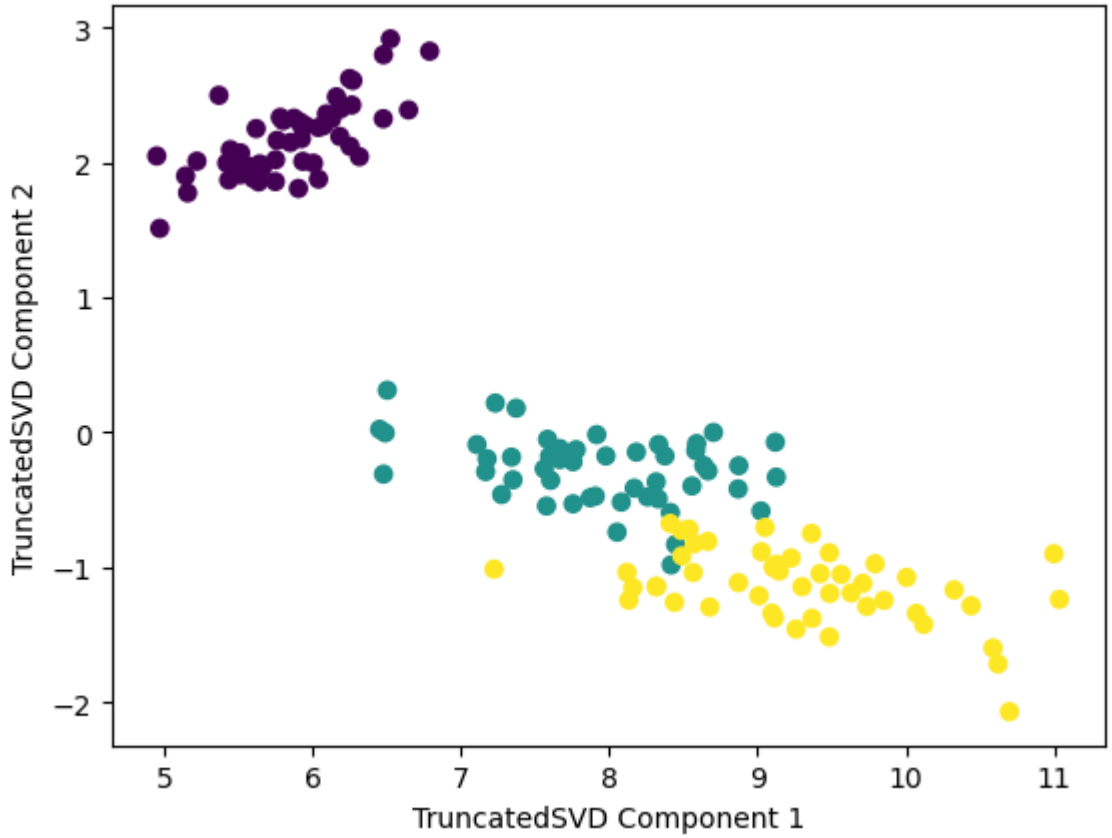
➢ 사이킷런 TruncatedSVD 클래스

```
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)

# Scatter plot 2차원으로 TruncatedSVD 변환 된 데이터 표현. 품종은 색깔로 구분
plt.scatter(x=iris_tsvd[:,0], y= iris_tsvd[:,1], c= iris.target)
plt.xlabel('TruncatedSVD Component 1')
plt.ylabel('TruncatedSVD Component 2')
```


1. 차원축소

❖SVD(Singular Value Decomposition)

➢ 사이킷런 TruncatedSVD 클래스



JEONJU UNIVERSITY

1. 차원축소

❖SVD(Singular Value Decomposition)

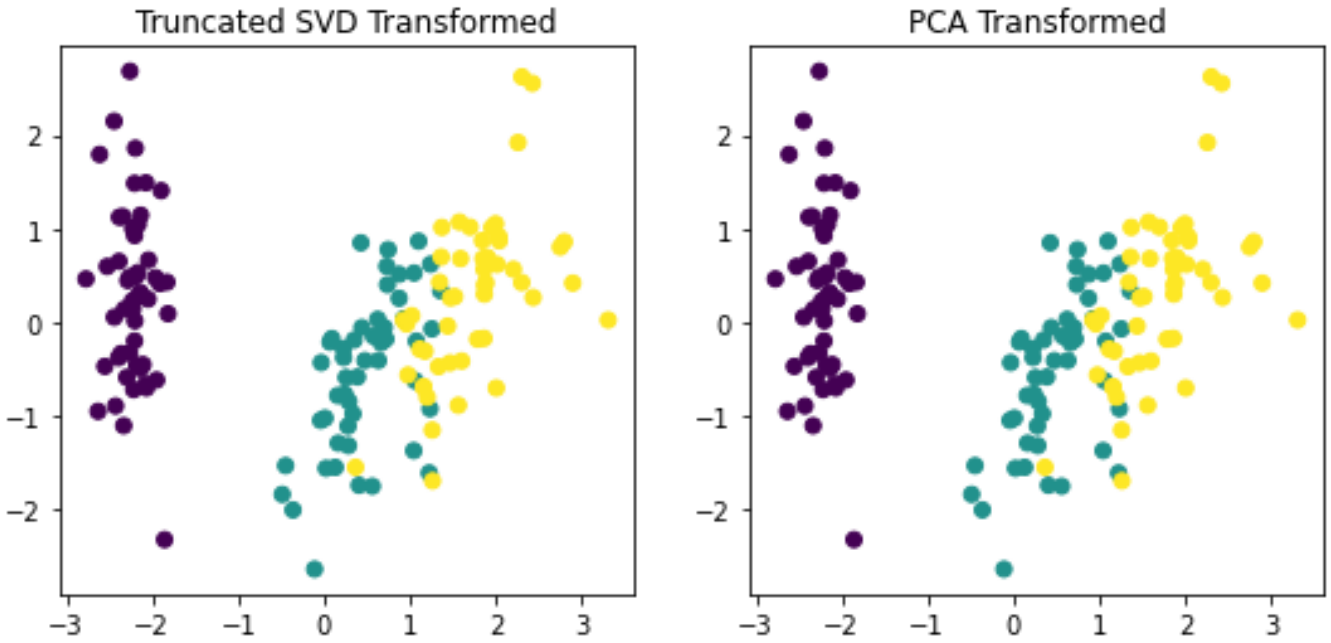
➢ 스케일링으로 변환한 뒤 TruncatedSVD와 PCA 클래스 변환

```
from sklearn.preprocessing import StandardScaler
# iris 데이터를 StandardScaler로 변환
scaler = StandardScaler()
iris_scaled = scaler.fit_transform(iris_ftrs)
# 스케일링된 데이터를 기반으로 TruncatedSVD 변환 수행
tsvd = TruncatedSVD(n_components=2)
tsvd.fit(iris_scaled)
iris_tsvd = tsvd.transform(iris_scaled)
# 스케일링된 데이터를 기반으로 PCA 변환 수행
pca = PCA(n_components=2)
pca.fit(iris_scaled)
iris_pca = pca.transform(iris_scaled)
# TruncatedSVD 변환 데이터를 왼쪽에, PCA변환 데이터를 오른쪽에 표현
fig, (ax1, ax2) = plt.subplots(figsize=(9,4), 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')
```


1. 차원축소

❖SVD(Singular Value Decomposition)

- ➢ 스케일링으로 변환한 뒤 TruncatedSVD와 PCA 클래스 변환
  - ✓ 거의 동일
  - ✓ PCA가 SVD 알고리즘으로 구현되었음을 의미

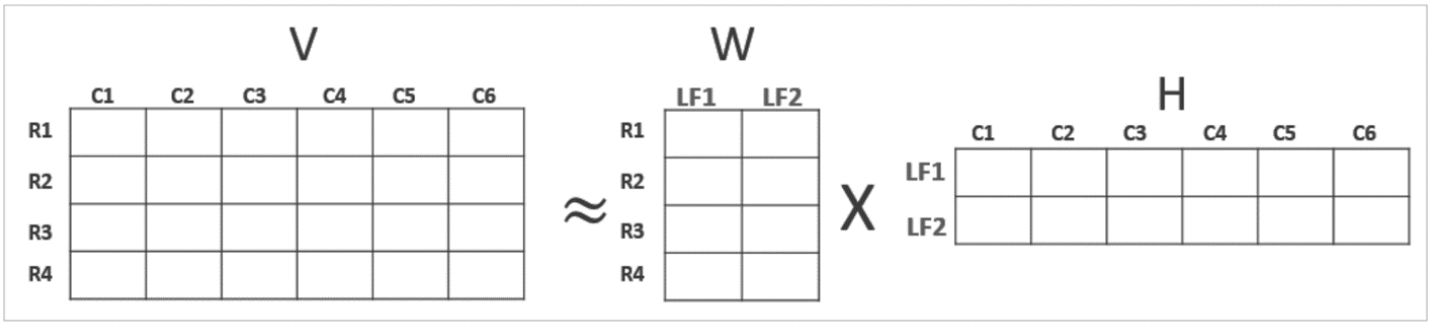


JEONJU UNIVERSITY

1. 차원축소

❖NMF(Non-Negative Matrix Factorization)

- ➢ 낮은 랭크를 통한 행렬 근사 방식의 변형
- ➢ 원본 행렬 내의 모든 원소 값이 모두 양수라는게 보장되면 더 간단하게 두개의 기반 양수 행렬로 분해될 수 있는 기법
  - ✓ 행렬분해를 하게 되면 일반적으로 길고 가는 행렬 W, 작고 넓은 행렬 H로 분해됨
  - ✓ 이렇게 분해된 행렬은 잠재 요소를 특성으로 가짐
  - ✓ 분해 행렬 W : 원본 행에 대해 이 잠재 요소의 값이 얼마나 되는지 대응
  - ✓ 분해 행렬 H : 이 잠재 요소가 원본 열로 어떻게 구성됐는지 나타내는 행렬



JEONJU UNIVERSITY

1. 차원축소

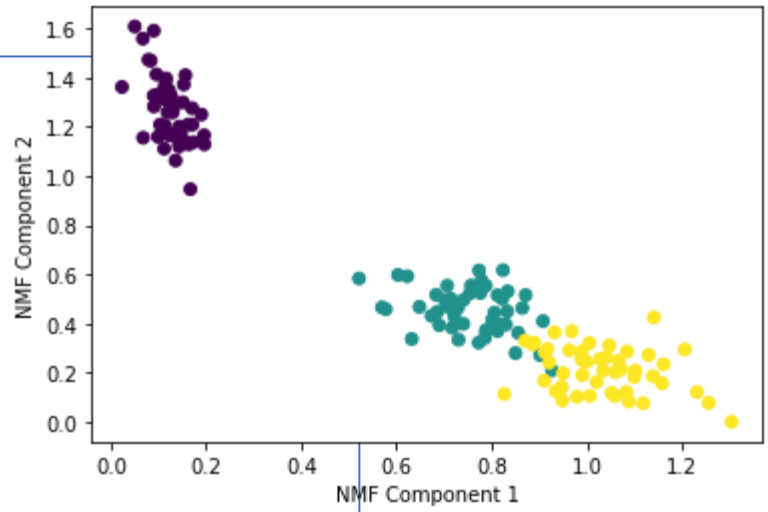
❖NMF(Non-Negative Matrix Factorization)

- NMF는 SVD와 유사하게 차원 축소를 통한 잠재 요소 도출로 이미지 변환 및 압축, 텍스트의 토픽 도출 등의 영역에서 사용됨
- 이미지 압축을 통한 패턴인식, 덱스트의 토픽모델링기법, 문서 유사도 및 클러스터링에 잘 사용됨
- 또한 영화추천과 같은 추천영역에 활발하게 적용됨
  - ✓ 사용자가 평가하지 않은 상품에 대한 잠재적인 요소를 추출해 이를 통해 평가 순위를 예측하고, 높은 순위로 예측된 상품을 추천해주는 방식
  - ✓ 이를 잠재요인 기반의 추천방식이라고 함

1. 차원축소

❖NMF(Non-Negative Matrix Factorization)

```
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')
```



Thank You !

JEONJU UNIVERSITY
The Place for Superstars