프로젝트 기반 데이터 과학자 양성과정(Data Science)
Machine Learning 및 분석실습

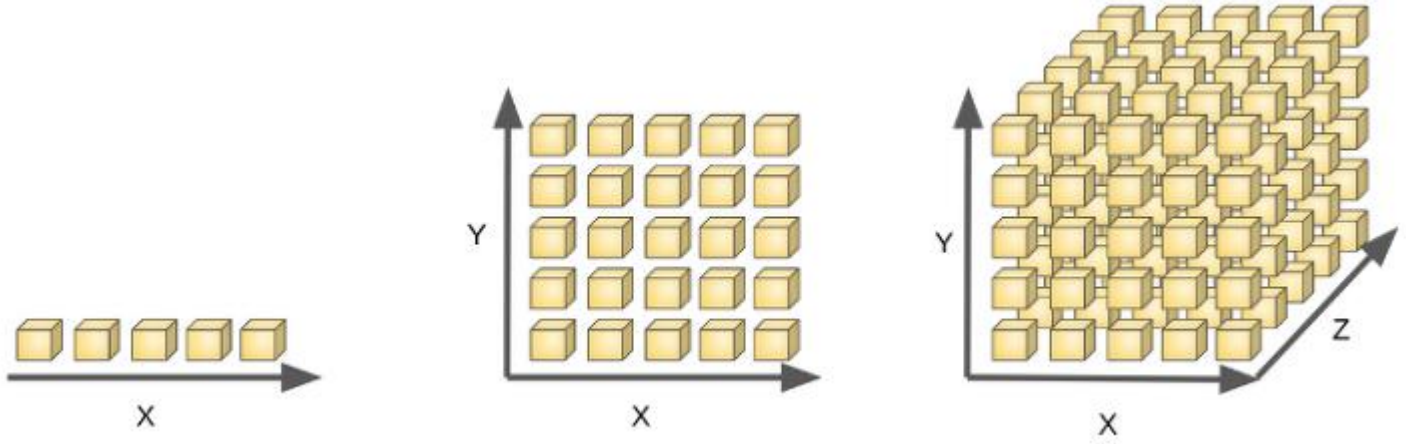
7주차
비지도학습
PCA
t-sne

강사 : 최영진

1. PCA

❖ 차원의 저주

- 데이터 셋의 특성(feature)가 많아지면, 각 특성인 하나의 차원(dimension) 또한 증가
- 데이터의 차원이 증가할수록 데이터 포인트 간의 거리 또한 증가하게 되므로, 이러한 데이터를 이용해 머신러닝 알고리즘을 학습시 모델이 복잡 → 오버피팅(overfitting) 위험
- 시각화→ 많은 수의 변수들에 대해서는 불가능
- 계산적인 병목현상→ 매우 많은 수의 변수들을 처리하는 것은 계산적으로 불가능
- Collinearity(공선성-매우 연관된 변수들) 또는 더 많은 변수들은 회귀 유형 모델에 문제를 발생



1. PCA

❖ PCA(Principal Component Analysis)

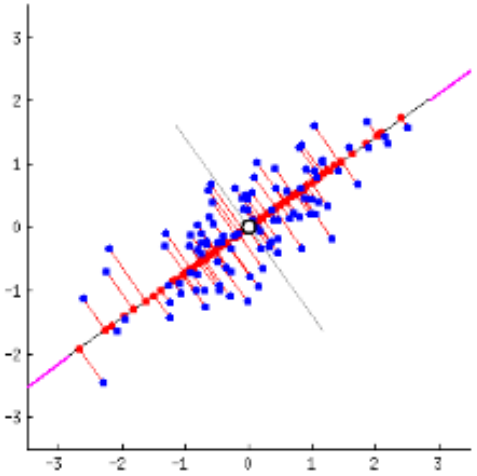
- 변수선택(selection)
  - 일부 주요한 변수만 선정하여 사용하는 방법
  - 선택한 변수 해석이 용이하고 필요한 변수가 있을 경우 사용

- 변수추출(extraction)
  - 기존 변수를 조합해 새로운 변수를 만드는 방법
  - 변수 해석의 어려움
  - Principal component analysis (PCA), Wavelets transforms, Autoencoder 등

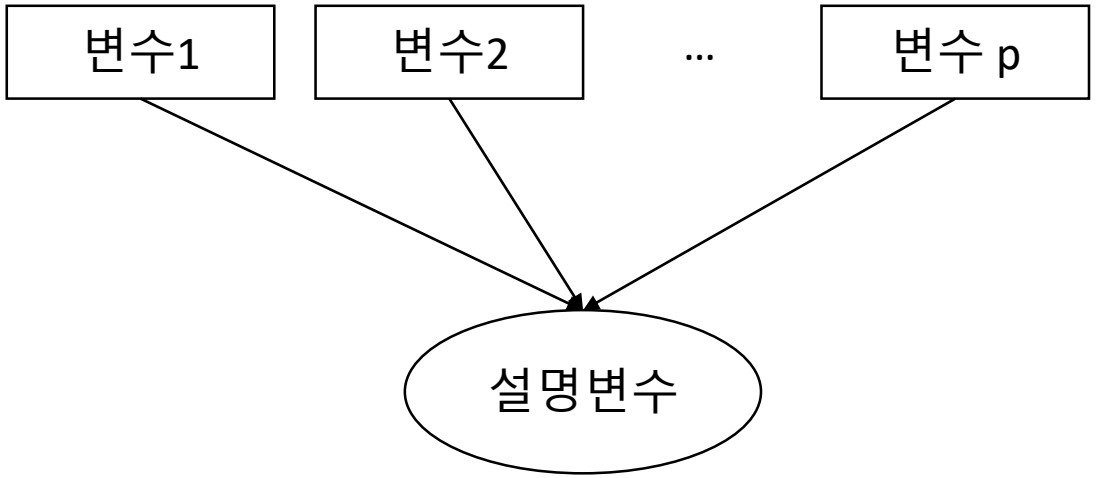
1. PCA

❖ PCA(Principal Component Analysis)

- PCA**는 Principal Component Analysis의 약자로 주성분분석이라고 함**
- **3개 이상의 다차원 데이터에서 Principal Component를 추출하여 특징(feature) 을 추출하거나 패턴(Pattern)을 찾는 통계 기반 분석 알고리즘**
- 원 데이터의 손실을 최소화하여 데이터를 설명할 수 있는 변수를 찾는 것이 목표
- 고차원 데이터를 저차원으로 사상 시켜 분산을 최대한 유지할 수 있는 방법



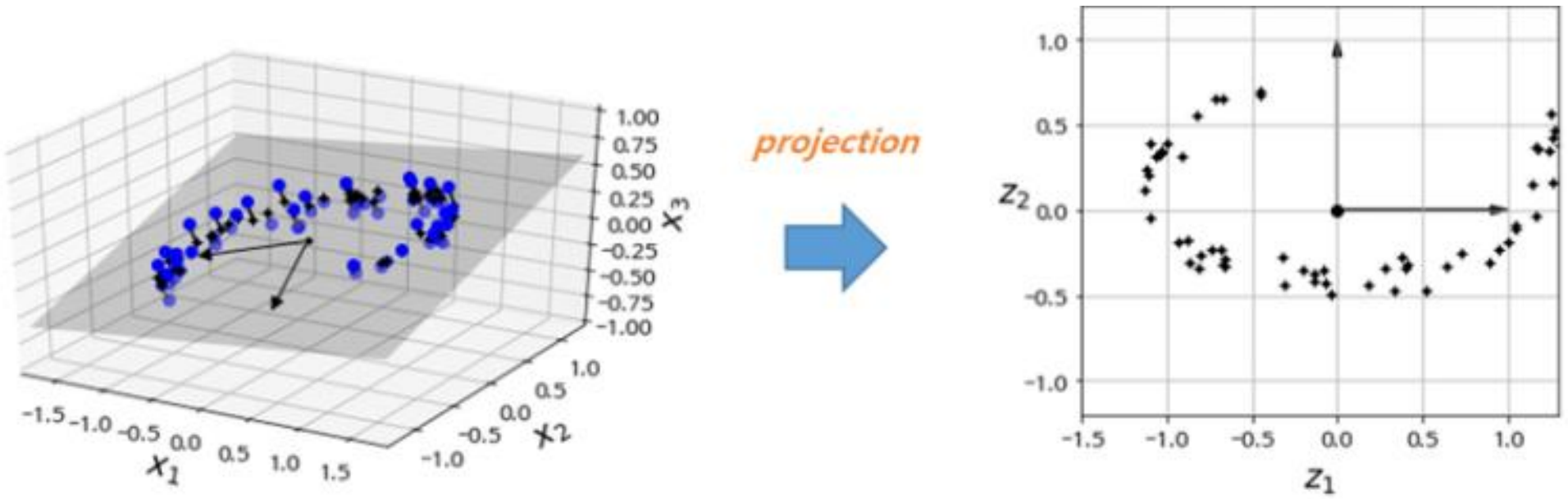
출처:https://deeplearning4j.org/kr/eigenvector



1. PCA

❖ 투영 (Projection)

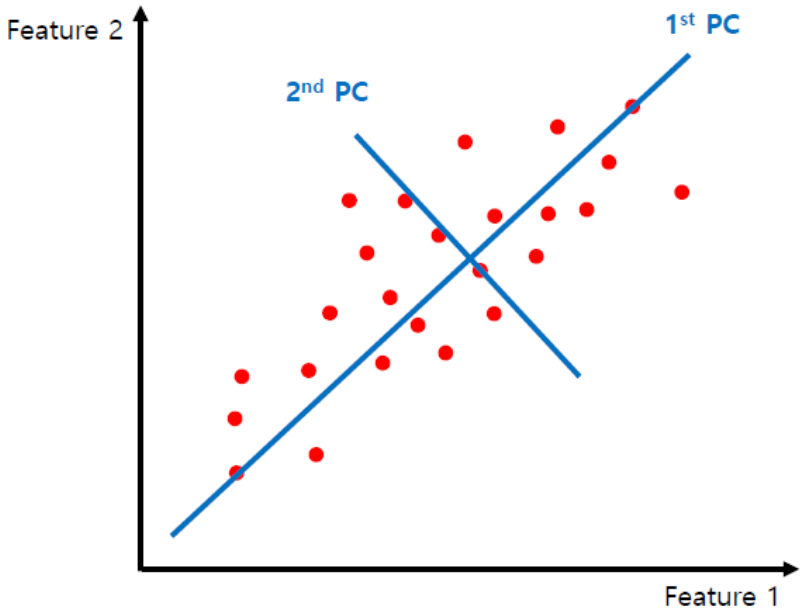
- 3차원 공간상의 데이터를 2차원 부분 공간으로 투영(projection)시켜 2차원 데이터셋에 투영



1. PCA

❖ PCA 단계

- 학습 데이터셋에서 분산이 최대인 축(axis)을 찾음
- 찾은 첫번째 축과 직교(orthogonal)하면서 분산이 최대인 두 번째 축을 찾음
- 같은 방법으로 데이터셋의 차원(특성 수)만큼의 축을 찾음

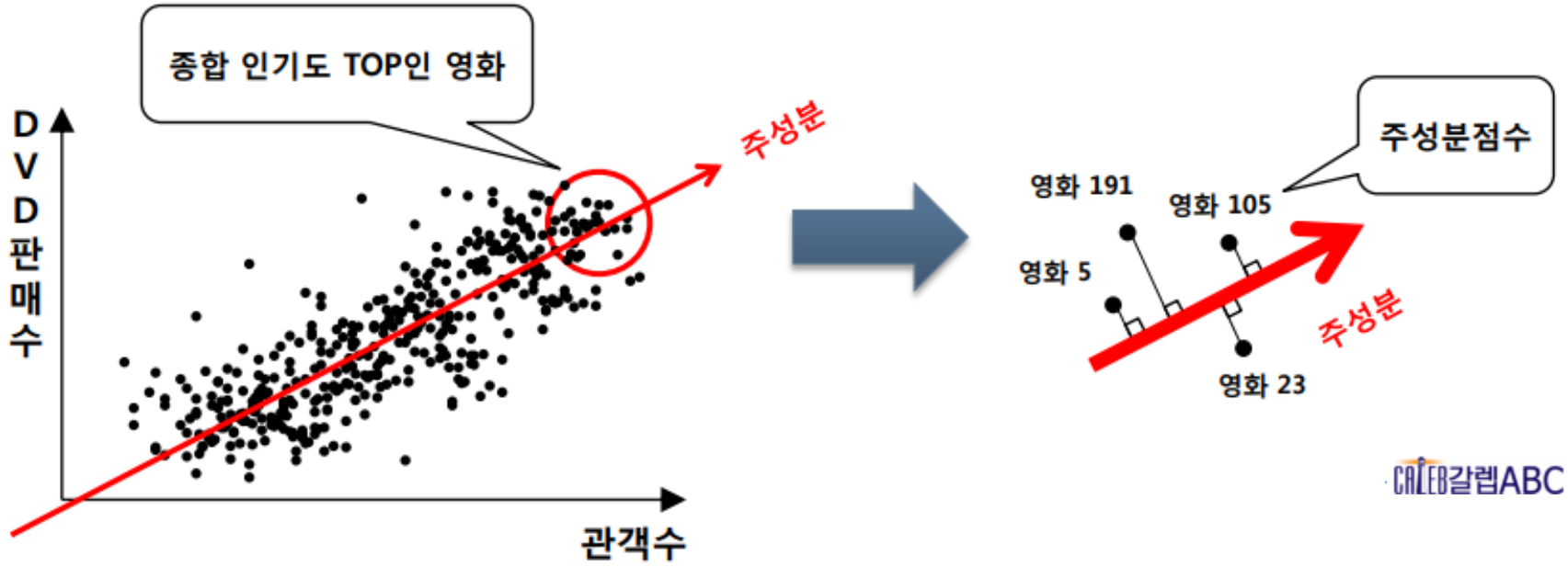


1. PCA

❖ PCA 예시

▪ 영화 관객수와 DVD 판매수

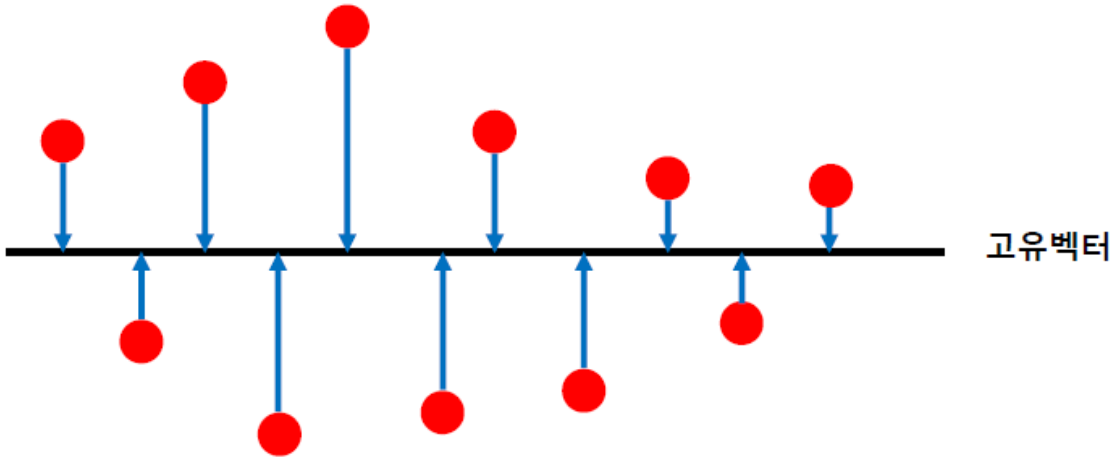
- 영화 105: 종합 인기도 TOP
- 축: 주성분, 좌표: 주성분점수

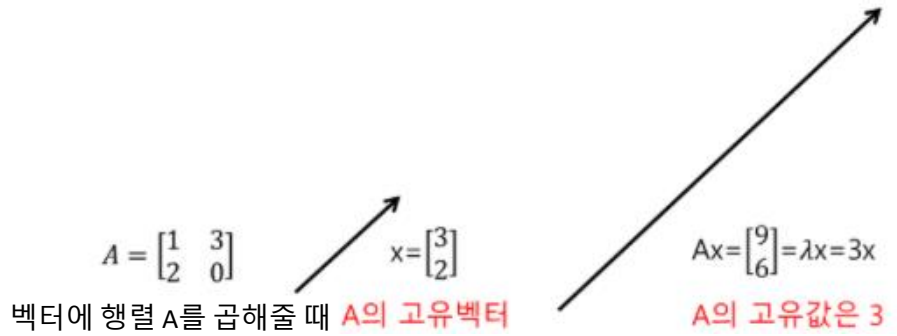


1. PCA

❖ 고유벡터(Eigenvectors)와 고유값(Eigenvalues)

- 공분산(covariance)을 분석하여 선형변환 행렬(linear transformation matrix)을 얻어 차원을 축소하는 방법
- 선형변환(Linear Transformation)은 벡터 공간에서 벡터 공간으로 가는 함수로, 벡터 공간의 성질을 보존하는, 선형성을 갖는 함수로 선형맵(Linear Map), 일차변환 의미
- 고유 벡터와 고유값을 이용
- 고유벡터는 데이터의 분포를 나타내는선, 고유값은 해당고유벡터에 대해 데이터가 분포하는 분산을 의미함





1. PCA

❖ 고유벡터(Eigenvectors)와 고유값(Eigenvalues)

- **공분산 계산**
  - 오리지널 공분산의 경우 자신의 분산을 제외하고도 다른 값들도 0이 아닌 값들을 보여줌
  - 즉, feature마다의 영향력으로 하나하나 독립적인 예측력을 지니도록 투영하는 작업

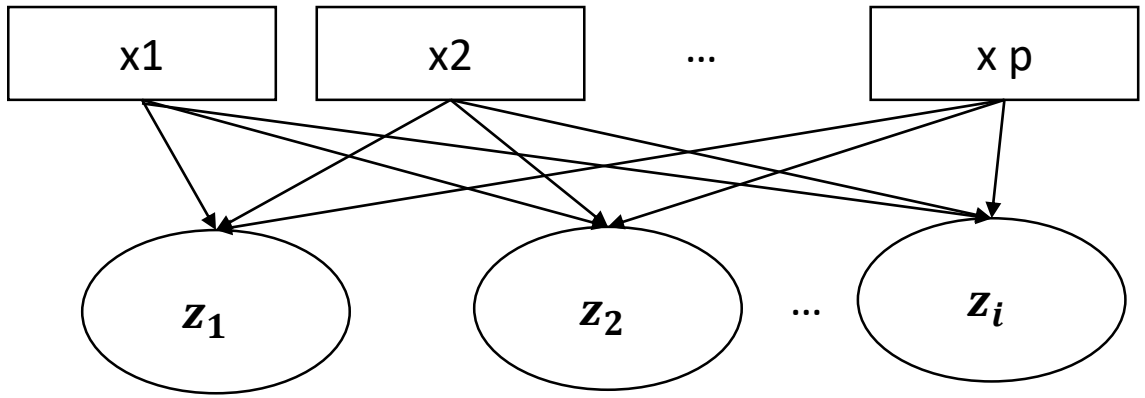
- **공분산으로부터 고유벡터, 고유값을 뽑아내고** 공분산행렬에서 고유값과 고유벡터를 계산
  - 공분산행렬을 선형변환을 통해서 서로 독립인 데이터를 만들 수 있음
  - 고유벡터는 각각의 벡터가 직교하기 때문에 무엇이든 투영할 수 있음
  - 고유값의 크기별로 내림차순으로 되어있기 때문에, 영향력에 대해 설명
  - 따라서 PC1부터 차례대로 기존의 행렬의 내적을 한다면, 축소된 차원이지만 정보를 얻게 됨

1. PCA

❖ 고유벡터(Eigenvectors)와 고유값(Eigenvalues)

- **선형변환 :** 변수가 p개, 관측치가 n개 있는 데이터 X(p x n)로 새로운 변수 z를 아래와 같이 만드는 과정
- **Z = X를 α_i 라는 새로운 축에 사영(projection)시킨 결과**

$$\vec{z_1} = \alpha_{11}\vec{x_1} + \alpha_{12}\vec{x_2} + \ldots + \alpha_{1p}\vec{x_p} = \vec{\alpha_1}^T X$$
$$\vec{z_2} = \alpha_{21}\vec{x_1} + \alpha_{22}\vec{x_2} + \ldots + \alpha_{2p}\vec{x_p} = \vec{\alpha_2}^T X$$
$$\ldots$$
$$\vec{z_p} = \alpha_{p1}\vec{x_1} + \alpha_{p2}\vec{x_2} + \ldots + \alpha_{pp}\vec{x_p} = \vec{\alpha_p}^T X$$



$$\Sigma = cov(X) = \frac{1}{n-1} XX^T \propto XX^T$$

i번째 주성분 z_i 큰 고유값 λi 과 고유벡터 αi

고유값 : 주성분의 분산 = 설명력
고유벡터 : 주성분 계수 = 변수의 중요도

첫 주성분은 데이터의 분산, 정보를 가장 많이 설명함.

$$\text{var}(C_1) \geq \text{var}(C_2) \geq \cdots \geq \text{var}(C_p) \Leftrightarrow \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p$$

1. PCA

❖ 공분산

- 데이터가 각 변수별로 평균이 0으로 맞춰져 있을 때 데이터의 공분산 행렬

$$\Sigma = cov(X) = \frac{1}{n-1} XX^T \propto XX^T$$

X의 차원수가 p x n이라면, 공분산행렬 Σ은 p x p 크기의 **정방행렬(square matrix)** , 또한 ΣT=Σ 인 **대칭행렬(symetric matrix)**

벡터가 공분산행렬 Σ의 고유벡터인 행렬을 A, 대각성분이 Σ의 고유값이고 대각성분을 제외한 요소값이 0인 행렬을 Λ

$$\Sigma A = \Lambda A$$
$$\Sigma = A \Lambda A^{-1}$$

$$\Sigma^T = (A^{-1})^T \Lambda A^T$$
$$= A \Lambda A^{-1} = \Sigma$$

$$\therefore A^{-1} = A^T$$
$$A^T A = I$$

공분산행렬 Σ의 서로 다른 고유벡터끼리는 서로 **직교(orthogonal)**함을 확인(I 는 단위행렬)

1. PCA

❖ 고유값과 새 변수의 분산

- Σ는 원데이터 X의 공분산 행렬이고, Σ의 고유값과 고유벡터를 각각 λ1, α1 가정

$$\vec{z_1} = \vec{\alpha_1}^T X$$
$$Var(\vec{z_1}) = \vec{\alpha_1}^T \Sigma \vec{\alpha_1}$$

$$\Sigma \vec{\alpha_1} = \lambda_1 \vec{\alpha_1}$$

$$\begin{aligned} Var(z_1) &= \vec{\alpha_1}^T \Sigma \vec{\alpha_1} \\ &= \vec{\alpha_1}^T \lambda_1 \vec{\alpha_1} \\ &= \lambda_1 \vec{\alpha_1}^T \vec{\alpha_1} \\ &= \lambda_1 \end{aligned}$$

$$\|\alpha\| = \vec{\alpha}^T \vec{\alpha} = 1$$

Σ의 i번째로 큰 고유값과 고유벡터를 각각 λi, αi

1. PCA

❖ PCA 계산 예제 - X의 공분산 행렬

- **Σ를 고유분해한 결과 행렬 A는 각각의 벡터가Σ의 고유벡터로 구성.**
- **대각행렬 Λ는 각각의 대각원소가 Σ의 고유값**
- λ1(2.7596)에 대응하는 고유벡터는 α1([0.5699,0.5765,−0.5855]T)

구분	n_1	n_2	n_3	n_4	n_5
p_1	-1.1930	-0.0370	-0.5919	0.3792	1.4427
p_2	-1.0300	-0.7647	-0.3257	1.0739	1.0464
p_3	1.5012	0.3540	-0.0910	-0.7140	-1.0502

$$\Sigma = cov(X) = \frac{1}{5-1} X' X'^{T}$$

```
[[ 1.00003489  0.84168247 -0.88400976]
 [ 0.84168247  0.99999019 -0.91324874]
 [-0.88400976 -0.91324874  0.99997862]]
```

1. PCA

❖ 고유값과 새 변수의 분산

- 고유벡터로 새로운 변수 z1, x1,x2,x3는 각각 X′의 행벡터 , 벡터들 앞에 붙는 계수는 해당하는 고유벡터의 요소값

구분	n_1	n_2	n_3	n_4	n_5
p_1	-1.1930	-0.0370	-0.5919	0.3792	1.4427
p_2	-1.0300	-0.7647	-0.3257	1.0739	1.0464
p_3	1.5012	0.3540	-0.0910	-0.7140	-1.0502

$$\Sigma A = A\Lambda$$

$$A = [\overrightarrow{\alpha_1} \quad \overrightarrow{\alpha_2} \quad \overrightarrow{\alpha_3}]$$
$$= \begin{bmatrix} 0.5699 & 0.7798 & 0.2590 \\ 0.5765 & -0.6041 & 0.5502 \\ -0.5855 & 0.1643 & 0.7938 \end{bmatrix}$$

$$\Lambda = \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix}$$
$$= \begin{bmatrix} 2.7596 & 0 & 0 \\ 0 & 0.1618 & 0 \\ 0 & 0 & 0.0786 \end{bmatrix}$$

$$\overrightarrow{z_1} = \overrightarrow{\alpha_1}^T X$$

$$= [0.5699 \quad 0.5765 \quad -0.5855] \begin{bmatrix} \overrightarrow{x_1} \\ \overrightarrow{x_2} \\ \overrightarrow{x_3} \end{bmatrix}$$

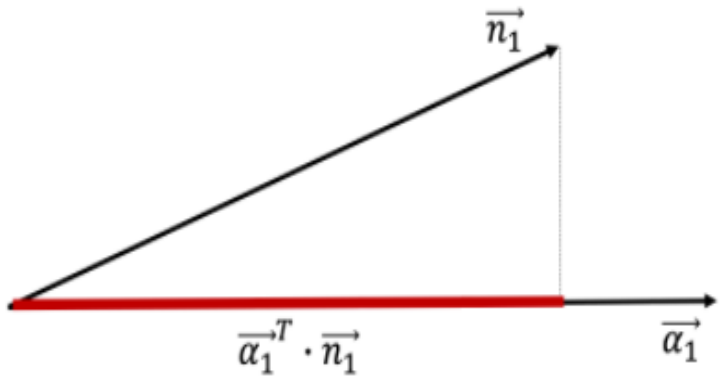
$$= 0.5699\overrightarrow{x_1} + 0.5765\overrightarrow{x_2} - 0.5855\overrightarrow{x_3}$$
$$= 0.5699\,[-1.1930 \quad -0.0370 \quad -0.5919 \quad 0.3792 \quad 1.4427]$$
$$+ 0.5765\,[-1.0300 \quad -0.7647 \quad -0.3257 \quad 1.0739 \quad 1.0464]$$
$$- 0.5855\,[1.5012 \quad 0.3540 \quad -0.0910 \quad -0.7140 \quad -1.0502]$$
$$= [-2.1527 \quad -0.6692 \quad -0.4718 \quad 1.2533 \quad 2.0404]$$

1. PCA

❖ 고유값과 새 변수의 분산

구분	n_1	n_2	n_3	n_4	n_5
p_1	-1.1930	-0.0370	-0.5919	0.3792	1.4427
p_2	-1.0300	-0.7647	-0.3257	1.0739	1.0464
p_3	1.5012	0.3540	-0.0910	-0.7140	-1.0502

- z1의 첫번째 요소(스칼라) -2.1527 의 의미
- 원데이터의 첫번째 데이터(n1)는 [-1.930, -1.0300, 1.5012]
- n1이 첫번째 고유벡터 α1과 내적 된 결과가 -2.1527
- 축에 해당하는 벡터가 유닛벡터일 때 벡터의 내적과 사영은 동일한 의미이므로 -2.1527은 n1을 α1라는 축에 사영해 α1에서 어디쯤 위치하는지 나타내는 값
- z1의 두번째 요소 -0.6692는 원데이터의 두번째 데이터 n2를 α1라는 축에 사영한 결과



1. PCA

❖ 고유값과 새 변수의 분산

$$Z = A^T X$$

$$\begin{bmatrix} \vec{z_1} \\ \vec{z_2} \\ \vec{z_3} \end{bmatrix} = \begin{bmatrix} \vec{\alpha_1}^T \\ \vec{\alpha_2}^T \\ \vec{\alpha_3}^T \end{bmatrix} X = \begin{bmatrix} -2.1527 & -0.6692 & -0.4718 & 1.2533 & 2.0404 \\ -0.0615 & 0.4912 & -0.2798 & -0.4703 & 0.3204 \\ 0.3160 & -0.1493 & -0.4047 & 0.1223 & 0.1157 \end{bmatrix}$$

$$cov(Z) = \begin{bmatrix} 2.7596 & 0 & 0 \\ 0 & 0.1618 & 0 \\ 0 & 0 & 0.0786 \end{bmatrix}$$

새 변수 z1,z2, z3사이에 공분산이 0이어서 **uncorrelated** 관계가 된 것을 확인
3개 주성분 가운데 첫번째 주성분(PC1)을 선택해도 원데이터의 설명력을 어느 정도 보존

$$\frac{\lambda_1}{\lambda_1 + \lambda_2 + \lambda_3} = \frac{2.7596}{2.7596 + 0.1618 + 0.0786} = 0.920$$

1. PCA

❖ PCA 실습

- **from sklearn.decomposition import PCA**
- **pca = PCA(n_components=2)**
- **pca.fit(X)**
- **pca.fit_transform(X)**

n_components	정수(PCA 수)
fit_transform()	특징행렬을 낮은 차원의 근사행렬로 변환
pca.components_	고유벡터
pca.explained_variance_ratio_	PCA가 설명하는 분산의 비율

1. PCA

❖ 고유벡터(Eigenvectors)와 고유값(Eigenvalues)

```
1  import numpy as np
2  import pandas as pd
3  data = {'x1' : [-1.1930,-0.0370,-0.5919,0.3792,1.4427],
4          'x2' : [-1.0300,-0.7647,-0.3257,1.0739,1.0464],
5          'x3' : [1.5012,0.3540,-0.0910,-0.7140,-1.0502]}
6
7
8
9
10 df=pd.DataFrame(data)
11
12 # df_0 = df.fillna(0)
13 print(df)
```

```
      x1      x2      x3
0 -1.1930 -1.0300  1.5012
1 -0.0370 -0.7647  0.3540
2 -0.5919 -0.3257 -0.0910
3  0.3792  1.0739 -0.7140
4  1.4427  1.0464 -1.0502
```

1. PCA

❖ 고유벡터(Eigenvectors)와 고유값(Eigenvalues)

```
# df_std = StandardScaler().fit_transform(df)
df_cov = (np.cov(df.T))
eig_vals, eig_vecs = np.linalg.eig(df_cov)

print('covariance ratio :₩n', df_cov)


```

```
covariance ratio :
 [[ 1.00003489  0.84168247 -0.88400976]
 [ 0.84168247  0.99999019 -0.91324874]
 [-0.88400976 -0.91324874  0.99997862]]
```

```
#고유벡터 계산
print(eig_vecs)
```

```
[[ 0.5699263   0.77981949  0.25897024]
 [ 0.57649781 -0.60405883  0.55023923]
 [-0.58552054  0.16430003  0.79383323]]
```

벡터의 방향만 반대일 뿐 주성분 벡터가 구성하는 축은 동일

```
from sklearn.decomposition import PCA


pca = PCA(n_components=3)
pca.fit(df)

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

```
[[-0.5699263  -0.57649781  0.58552054]
 [ 0.77981949 -0.60405883  0.16430003]
 [-0.25897024 -0.55023923 -0.79383323]]
```

1. PCA

❖ 고유벡터(Eigenvectors)와 고유값(Eigenvalues)

```
1  #pc1 계산
2  PC1 = df.dot(np.reshape(eig_vecs.T[0], (3, 1)))
3  print(PC1)
```

```
          0
0 -2.152698
1 -0.669209
2 -0.471822
3  1.253279
4  2.040394
```

```
1  eig_vals_sum = np.sum(eig_vals)
2  for eig_val in eig_vals:
3      print ("variance : " + str(eig_val/eig_vals_sum))
4
```

```
variance : 0.9198722829162319
variance : 0.05393469393283531
variance : 0.026193023150932717
```

1. PCA

❖ 고유벡터(Eigenvectors)와 고유값(Eigenvalues)

```python
import numpy as np
import numpy.linalg as linalg
from sklearn.preprocessing import StandardScaler
from numpy import genfromtxt
from sklearn import datasets

iris = datasets.load_iris()
iris_data = iris['data']
y = iris['target']

iris_std = StandardScaler().fit_transform(iris_data)
iris_cov = (np.cov(iris_std.T))
eig_vals, eig_vecs = np.linalg.eig(iris_cov)

print('covariance ratio :₩n', iris_cov)
eig_vals_sum = np.sum(eig_vals)
for eig_val in eig_vals:
    print ("variance : " + str(eig_val/eig_vals_sum))

#고유벡터 계산
# PC1 = iris_std.dot(np.reshape(eig_vecs.T[0], (4, 1)))

```

```
covariance ratio :
 [[ 1.00671141 -0.11835884  0.87760447  0.82343066]
 [-0.11835884  1.00671141 -0.43131554 -0.36858315]
 [ 0.87760447 -0.43131554  1.00671141  0.96932762]
 [ 0.82343066 -0.36858315  0.96932762  1.00671141]]
variance : 0.7296244541329983
variance : 0.22850761786701818
variance : 0.03668921889282866
variance : 0.005178709107154747
```

1. PCA

❖ PCA(Principal Component Analysis)

- 장점
  - 고차원의 데이터를 저차원으로 변환하는 차원축소 방법으로 시각화에 용이함
  - PCA로 찾은 주성분은 전체 데이터를 설명함
  - 변수의 수가 많아 불필요한 변수가 존재할 때, 사용
  - 영상인식, 통계 데이터 분석(주성분 찾기), 데이터 압축(차원감소), 노이즈 제거 등 다양한 활용

- 단점
  - 2~3개의 변수로 설명하고자 하여 설명령이 떨어지는 경우가 다수발생
  - 선형변환을 이용하기 때문에 비선형 특성을 가진 데이터에 대해서는 데이터의 특성을 잘 추출하지 못하는 한계

1. PCA

❖ PCA 실습

```python
import numpy as np
import numpy.linalg as linalg
from sklearn.preprocessing import StandardScaler
from numpy import genfromtxt
from sklearn import datasets

iris = datasets.load_iris()
iris_data = iris['data']
y = iris['target']

iris_std = StandardScaler().fit_transform(iris_data)
iris_cov = (np.cov(iris_std.T))
eig_vals, eig_vecs = np.linalg.eig(iris_cov)

print('covariance ratio :₩n', iris_cov)
eig_vals_sum = np.sum(eig_vals)
for eig_val in eig_vals:
    print ("variance : " + str(eig_val/eig_vals_sum))

```

```
covariance ratio :
 [[ 1.00671141 -0.11835884  0.87760447  0.82343066]
 [-0.11835884  1.00671141 -0.43131554 -0.36858315]
 [ 0.87760447 -0.43131554  1.00671141  0.96932762]
 [ 0.82343066 -0.36858315  0.96932762  1.00671141]]
variance : 0.7296244541329983
variance : 0.22850761786701818
variance : 0.03668921889282866
variance : 0.005178709107154747
```

1. PCA

❖ PCA 실습

```
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()

scaler = StandardScaler()
scaler.fit(cancer.data)
X_scaled = scaler.transform(cancer.data)
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()

scaler = StandardScaler()
scaler.fit(cancer.data)
X_scaled = scaler.transform(cancer.data)
```

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)

pca.fit(X_scaled)

# 처음 두 개의 주성분을 사용해 데이터를 변환합니다
X_pca = pca.transform(X_scaled)
print("원본 데이터 형태:", str(X_scaled.shape))
print("축소된 데이터 형태:", str(X_pca.shape))
```

```
원본 데이터 형태: (569, 30)
축소된 데이터 형태: (569, 2)
```

1. PCA

❖ PCA 실습

```
print(pca.explained_variance_ratio_)
# 본 데이터의 경우 두 개의 주성분이 전체 분산의 약 63%를 설명

```

```
[0.44272026 0.18971182]
```

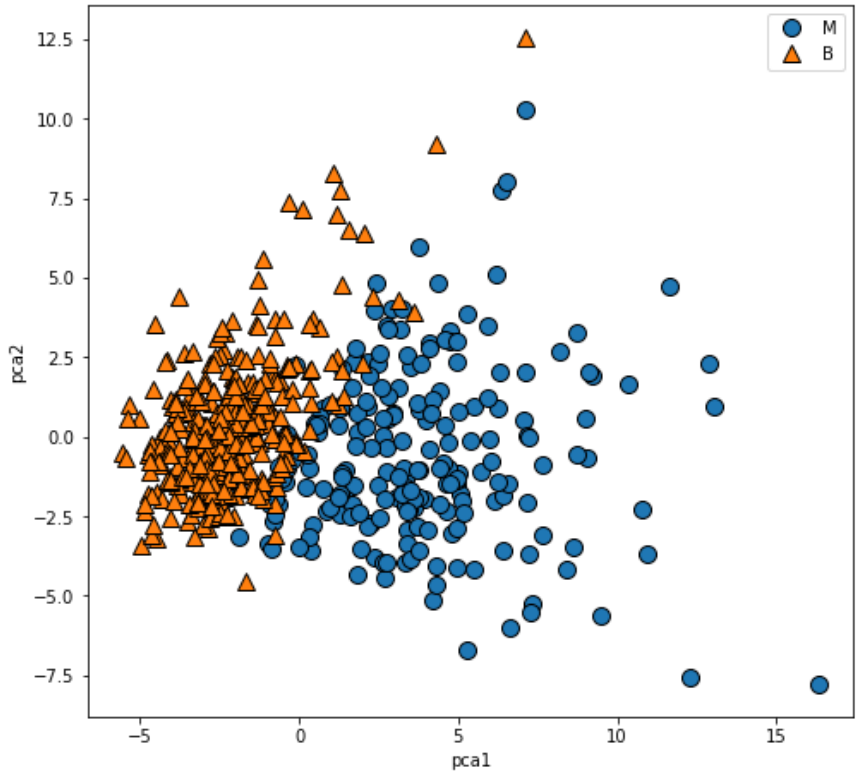
```
print("PCA 주성분:", pca.components_)

```

```
PCA 주성분: [[ 0.21890244  0.10372458  0.22753729  0.22099499  0.14258969  0.23928535
   0.25840048  0.26085376  0.13816696  0.06436335  0.20597878  0.01742803
   0.21132592  0.20286964  0.01453145  0.17039345  0.15358979  0.1834174
   0.04249842  0.10256832  0.22799663  0.10446933  0.23663968  0.22487053
   0.12795256  0.21009588  0.22876753  0.25088597  0.12290456  0.13178394]
 [-0.23385713 -0.05970609 -0.21518136 -0.23107671  0.18611302  0.15189161
   0.06016536 -0.0347675   0.19034877  0.36657547 -0.10555215  0.08997968
  -0.08945723 -0.15229263  0.20443045  0.2327159   0.19720728  0.13032156
   0.183848    0.28009203 -0.21986638 -0.0454673  -0.19987843 -0.21935186
   0.17230435  0.14359317  0.09796411 -0.00825724  0.14188335  0.27533947]]
```

1. PCA

❖ PCA 실습

```
import matplotlib.pyplot as plt
import numpy as np
import mglearn

# 클래스를 색깔로 구분하여 처음 두 개의 주성분을 그래프로 나타냅니다.
plt.figure(figsize=(8, 8))
mglearn.discrete_scatter(X_pca[:, 0], X_pca[:, 1], cancer.target)
plt.legend(["M", "B"], loc="best")
plt.gca().set_aspect("equal")
plt.xlabel("pca1")
plt.ylabel("pca2")
```

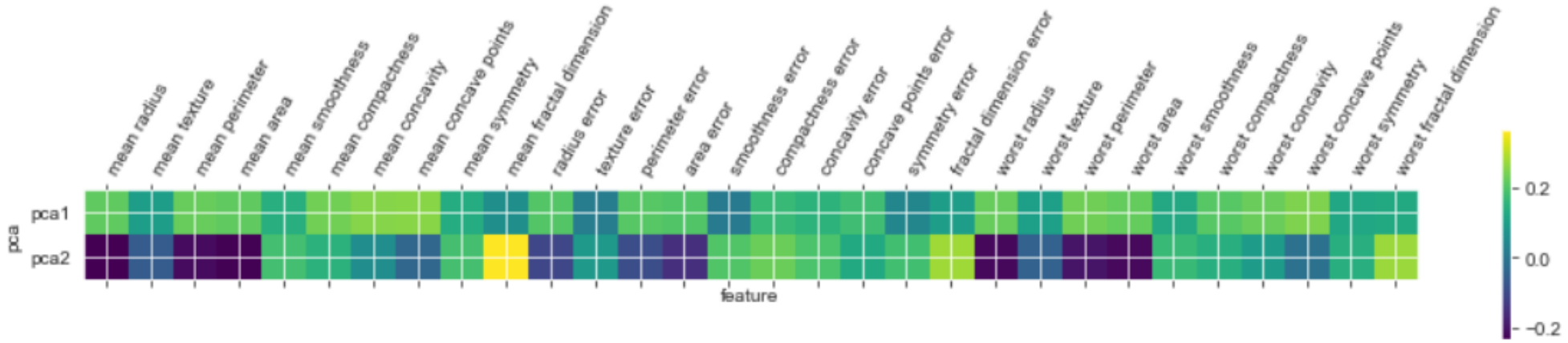


1. PCA

❖ PCA 실습

```
plt.matshow(pca.components_, cmap='viridis')
plt.yticks([0, 1], ["pca1", "pca2"])
plt.colorbar()
plt.xticks(range(len(cancer.feature_names)),
           cancer.feature_names, rotation=60, ha='left')
plt.xlabel("feature")
plt.ylabel("pca")

```

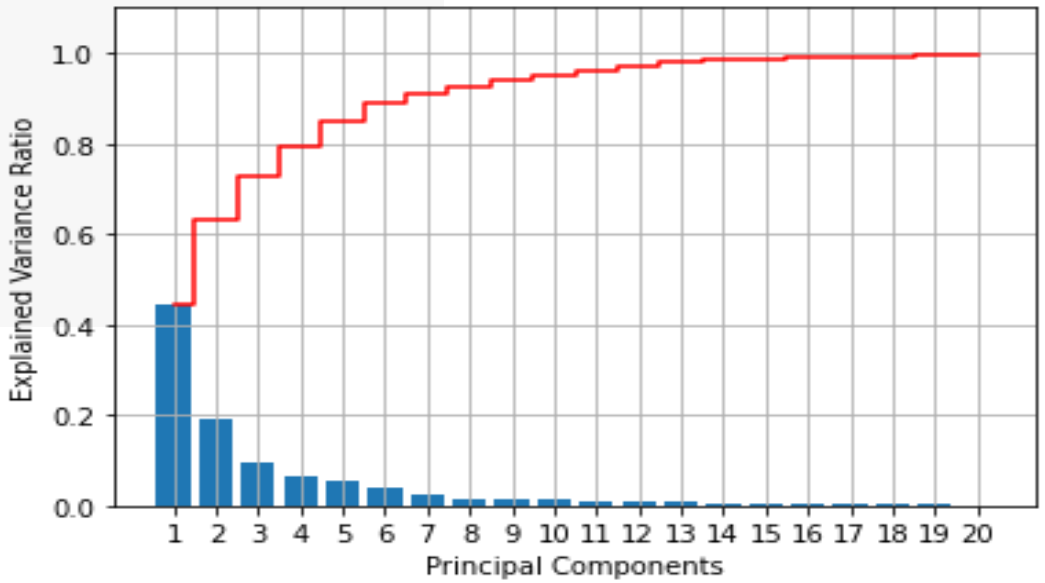
```
Text(0, 0.5, 'pca')
```



1. PCA

❖ PCA 실습

```
pca = PCA(n_components=20)
pca.fit(X_scaled)

X_pca = pca.transform(X_scaled)
explained_variance_ratio = pca.explained_variance_ratio_

def variance_ratio_plot(explained_variance_ratio):
    x_axis = range(1, len(explained_variance_ratio)+1)
    plt.bar(x_axis, explained_variance_ratio, align = 'center')
    plt.step(x_axis, np.cumsum(explained_variance_ratio),where = 'mid', color='red')
    plt.ylim(0, 1.1)
    plt.xticks(x_axis)
    plt.xlabel('Principal Components')
    plt.ylabel('Explained Variance Ratio')
    plt.grid()
    plt.show()

variance_ratio_plot(explained_variance_ratio)
```



1. PCA

❖ PCA 실습

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
from sklearn.datasets import load_iris

iris = load_iris()
```

```
from sklearn.decomposition import PCA

scaler = StandardScaler()
scaler.fit(iris.data)
X_scaled = scaler.transform(iris.data)

pca = PCA(n_components=2)
pca.fit(X_scaled)

print(pca.components_)
print(pca.explained_variance_ratio_)
# 본 데이터의 경우 두 개의 주성분이 전체 분산의
```

```
[[ 0.52106591 -0.26934744  0.5804131   0.56485654]
 [ 0.37741762  0.92329566  0.02449161  0.06694199]]
[0.72962445 0.22850762]
```

```
from sklearn.decomposition import PCA

pca_3 = PCA(n_components=3)
pca_3.fit(X_scaled)

print(pca_3.components_)
print(pca_3.explained_variance_ratio_)
# 본 데이터의 경우 두 개의 주성분이 전체 분산의 약 99%를 설명
```

```
[[ 0.52106591 -0.26934744  0.5804131   0.56485654]
 [ 0.37741762  0.92329566  0.02449161  0.06694199]
 [-0.71956635  0.24438178  0.14212637  0.63427274]]
[0.72962445 0.22850762 0.03668922]
```

1. PCA

❖ PCA 실습

```
1  X_pca = pca.transform(X_scaled)
2  X_pca.shape
```

```
(150, 2)
```

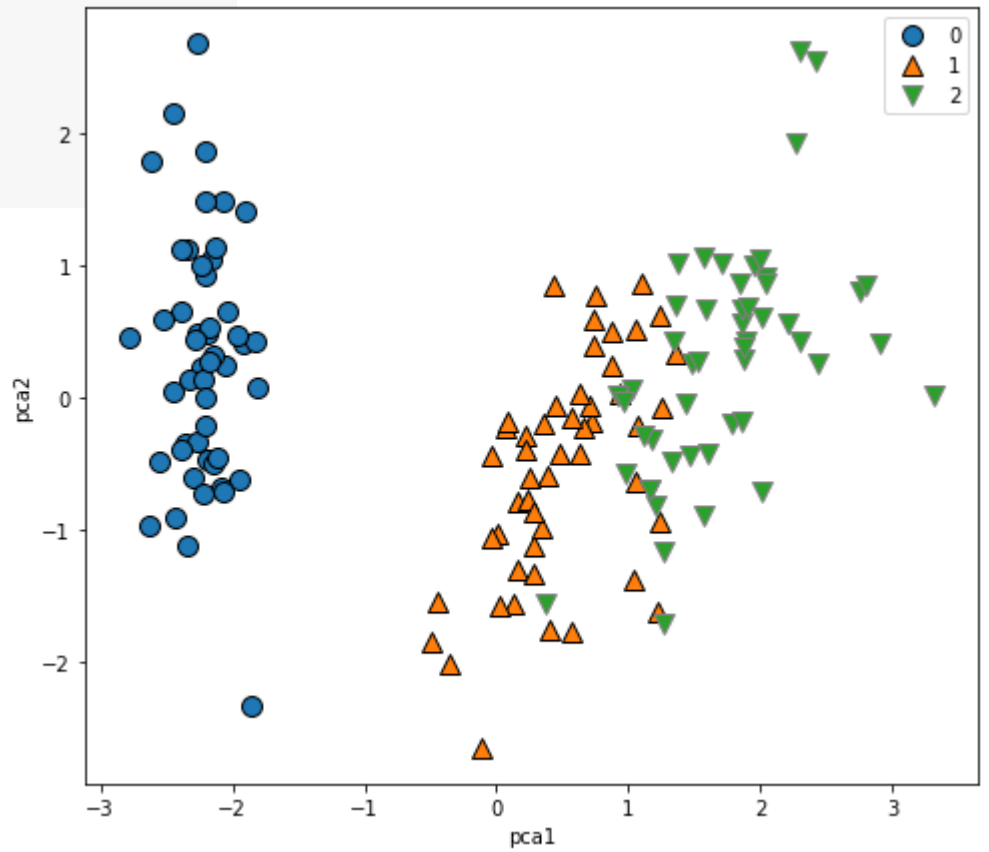
```
1  principalDf = pd.DataFrame(data=X_pca, columns = ['principal component1', 'principal component2'])
2  principalDf.head()
3
```

	principal component1	principal component2
0	-2.264703	0.480027
1	-2.080961	-0.674134
2	-2.364229	-0.341908
3	-2.299384	-0.597395
4	-2.389842	0.646835

1. PCA

❖ PCA 실습

```
import matplotlib.pyplot as plt

# 클래스를 색깔로 구분하여 처음 두 개의 주성분을 그래프로 나타냅니다.
plt.figure(figsize=(8, 8))
mglearn.discrete_scatter(X_pca[:, 0], X_pca[:, 1], iris.target)
plt.legend(loc="best")
plt.gca().set_aspect("equal")
plt.xlabel("pca1")
plt.ylabel("pca2")
```



2. t-sne

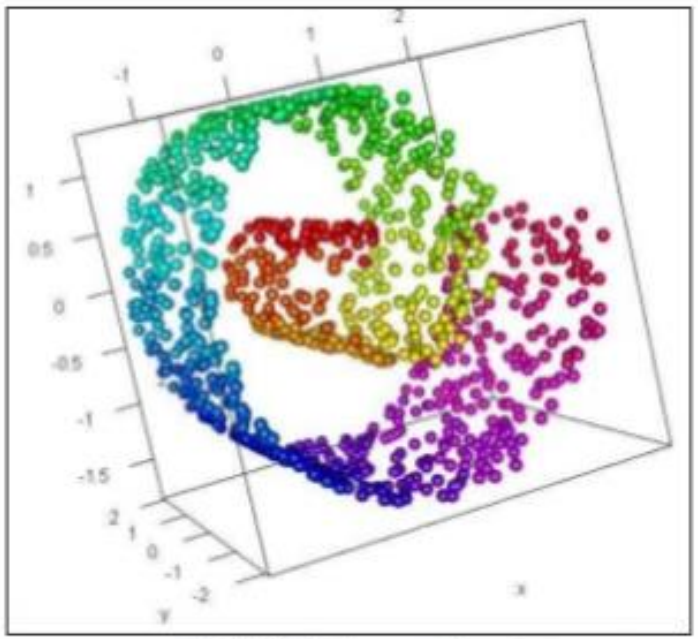
❖ t-Stochastic Nearest Neighbor

- 차원 축소, 임베딩 알고리즘들은 이전에도 제안
- 1960 년대에 제안되었던 Multi-Dimensional Scaling (MDS) 부터
  2000 년에 제안된 ISOMAP 와 Locally Linear Embedding (LLE) 모두 고차원의 정보를 저차원으로 축소
- Pincipal Components Analysis (PCA) 역시 고차원에서의 거리 정보를 보존하여 저차원으로 차원을 변환

2. t-sne

❖ t-Stochastic Nearest Neighbor

- maniford learning 방식 – 복잡한 매핑을 연결하여 더 나은 시각화(고차원의 데이터를 저차원상에 나타냄)
- 대부분 label이 없는 비지도 학습에 사용
- 3개 이상(3차원) 특성을 뽑는 경우는 없음



다양체의 예 : 스위스 롤

2. t-sne

❖ t-Stochastic Nearest Neighbor(t-SNE)

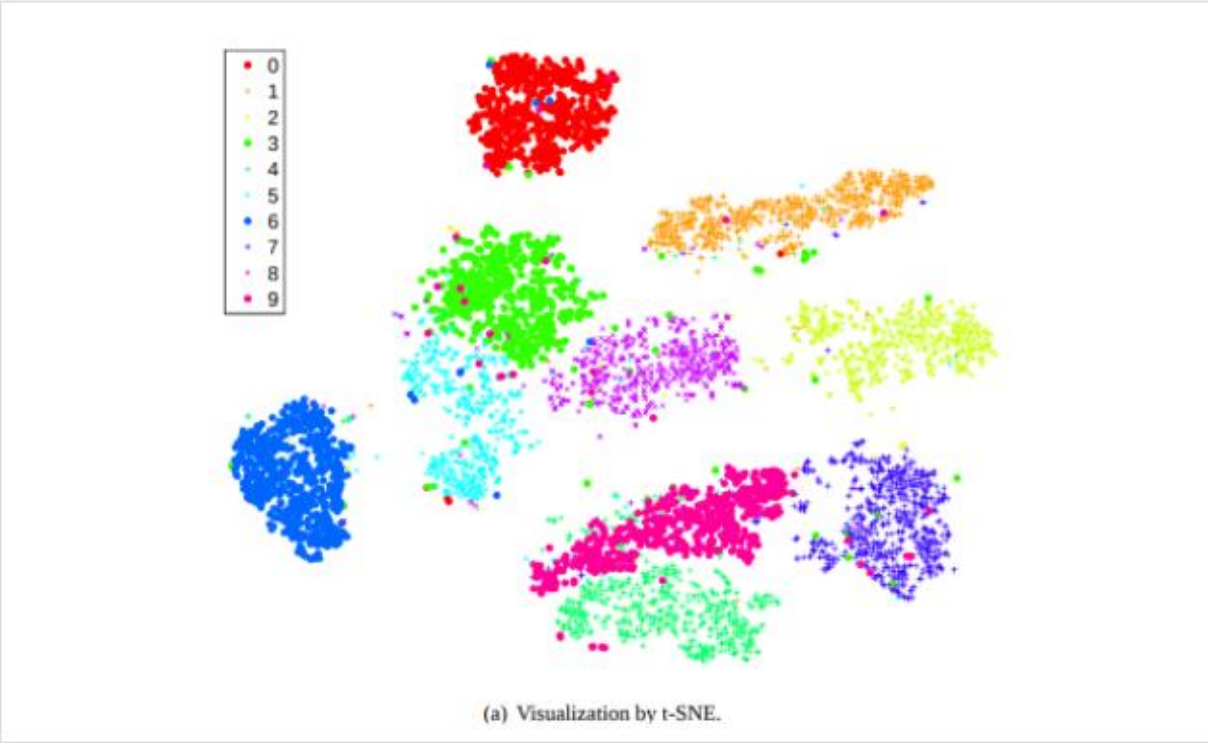
- 고차원의 벡터로 표현되는 데이터 간의 neighbor structure 를 보존하는 2 차원의 embedding vector 를 학습함으로써, 고차원의 데이터를 2 차원의 지도로 표현
- t-SNE 는 벡터 시각화를 위한 다른 알고리즘들보다 안정적인 임베딩 학습 결과
- t-SNE 가 데이터 간 유클리디언 거리를 stochastic probability 로 변환하고 확률값이 높으면 포인트가 가깝고, 낮으면 포인트가 멀다.
- stochastic probability 는 perplexity 에 의하여 조절
- perplexity 를 이용하여 데이터 간 거리를 정의하는 방식 자체가 안정성이 있음

데이터의 유사도 $$p_{j|i} = \frac{exp(-|x_i - x_j|^2/2\sigma_i^2)}{\sum_{k \neq i} exp(-|x_i - x_k|^2/2\sigma_i^2)}$$

2. t-sne

❖ t-Stochastic Nearest Neighbor

- 차원 공간에서 유사한 두 벡터가 2 차원 공간에서도 유사하도록 원 공간에서의 점들 간 유사도를 보존하면서 차원축소
- **stochastic**이란 이름이 붙은 이유는 거리 정보를 **확률적**으로 계산

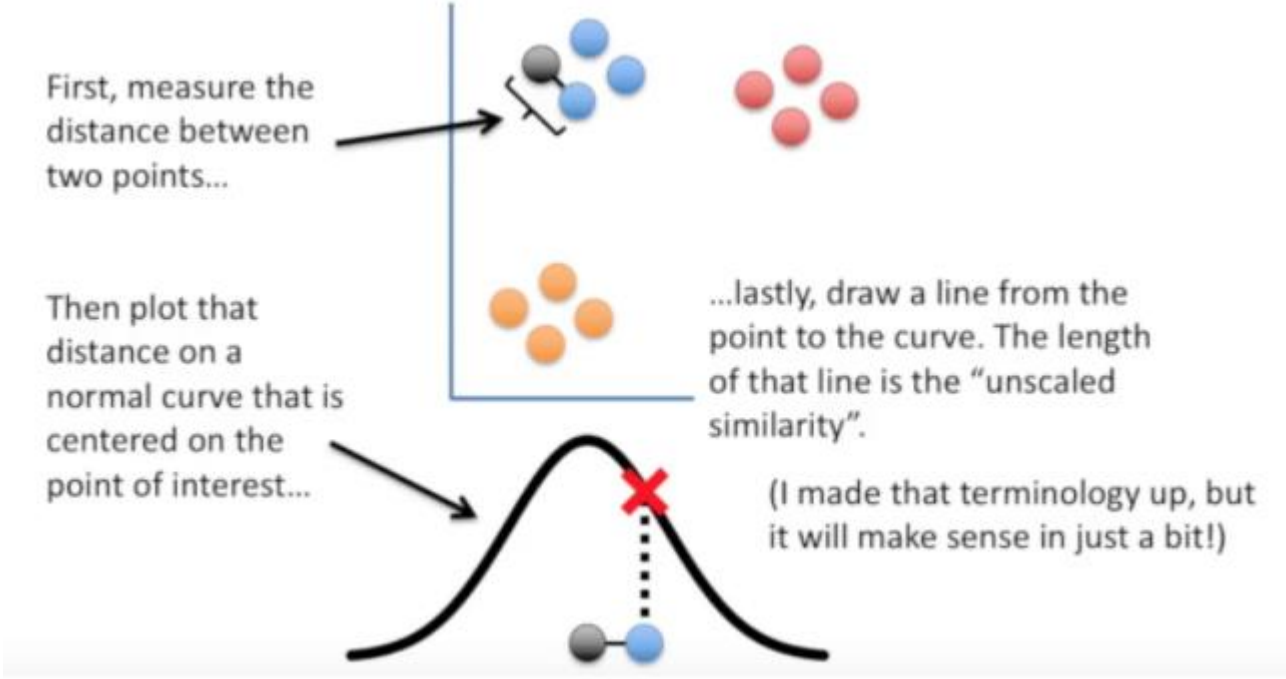


(a) Visualization by t-SNE.

2. t-sne

❖ t-Stochastic Nearest Neighbor

- 검정색점을 선택하고, 이 점에서 부터 다른 점까지의 거리를 측정
- 각 데이터 포인트를 2차원에 무작위로 표현
- 원본 특성 공간에서 가까운 포인트는 가깝게, 멀리 떨어진 포인트는 더 멀어지게 만듦
- 멀리 떨어진 포인트와 거리 보존보다 가까이 있는 포인트에 비중을 둠



2. t-sne

❖ t-Stochastic Nearest Neighbor

- 장점
  - PCA 처럼 군집이 중복되지 않는 장점
- 단점
  - 매번 계산할 때 마다 축의 위치가 바뀌어서, 다른 모양
  - 데이터의 군집성과 같은 특성들은 유지 되기 때문에 시각화를 통한 데이타 분석에는 유용
  - 매번 값이 바뀌는 특성으로 인하여, 머신러닝 모델의 학습 feature로 사용하기는 다소 어려운점

2. t-sne

❖ t-sne 실습

- from sklearn.manifold import TSNE
- tsne = TSNE(n_components=2)
- y = tsne.fit_transform(x)
- xs = transformed[:,0]
- ys = transformed[:,1]
- plt.scatter(xs,ys,c=labels)
- plt.show()

2. t-sne

❖ t-sne 실습

```
from sklearn.manifold import TSNE

TSNE(n_components=2, perplexity=30.0, early_exaggeration=12.0,
     learning_rate=200.0, n_iter=1000, n_iter_without_progress=300,
     min_grad_norm=1e-07, metric='euclidean', init='random', verbose=0,
     random_state=None, method='barnes_hut', angle=0.5)
```

parameter	note
n_components	임베딩 공간의 차원
perplexity	σi 의 기준, 학습에 영향을 주는 점들의 개수를 조절
metric	원 공간에서의 두 점간의 거리 척도
method	원 공간 데이터의 인덱싱 방식

2. t-sne

❖ t-sne 실습

```
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()

scaler = StandardScaler()
scaler.fit(cancer.data)
X_scaled = scaler.transform(cancer.data)
```

```
from sklearn.manifold import TSNE
tsne = TSNE(n_components = 2, random_state = 0)

```

```
tsne_obj = tsne.fit_transform(X_scaled)
```

```
tsne_df = pd.DataFrame({'X' : tsne_obj[:,0],
                        'Y' : tsne_obj[:,1],
                         'classification' : cancer.target
                        })
```

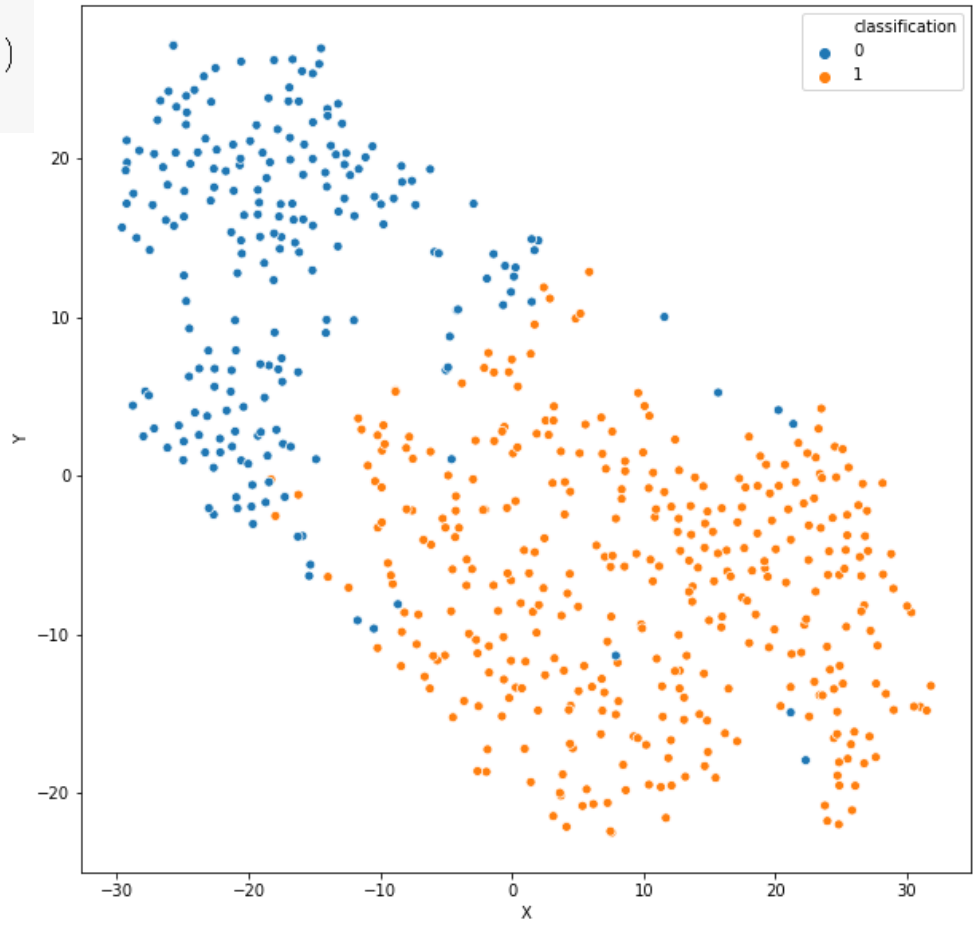
```
tsne_df.head()
```

	X	Y	classification
0	-26.271820	16.096264	0
1	-12.890461	22.179155	0
2	-22.607653	18.168743	0
3	-27.978006	2.484692	0
4	-15.115546	15.759376	0

2. t-sne

❖ t-sne 실습

```
import seaborn as sns
plt.figure(figsize = (10,10))
sns.scatterplot(x = "X", y = 'Y', hue = 'classification', legend = 'full', data = tsne_df)
plt.show()
```



2. t-sne

❖ t-sne 실습

```
from sklearn.datasets import load_iris

iris = load_iris()
```

```
from sklearn.decomposition import PCA

scaler = StandardScaler()
scaler.fit(iris.data)
X_scaled = scaler.transform(iris.data)

```

2. t-sne

❖ t-sne 실습

```
from sklearn.manifold import TSNE

tsne = TSNE(n_components=2, n_iter=1000, random_state=42)
points = tsne.fit_transform(X_scaled)
```

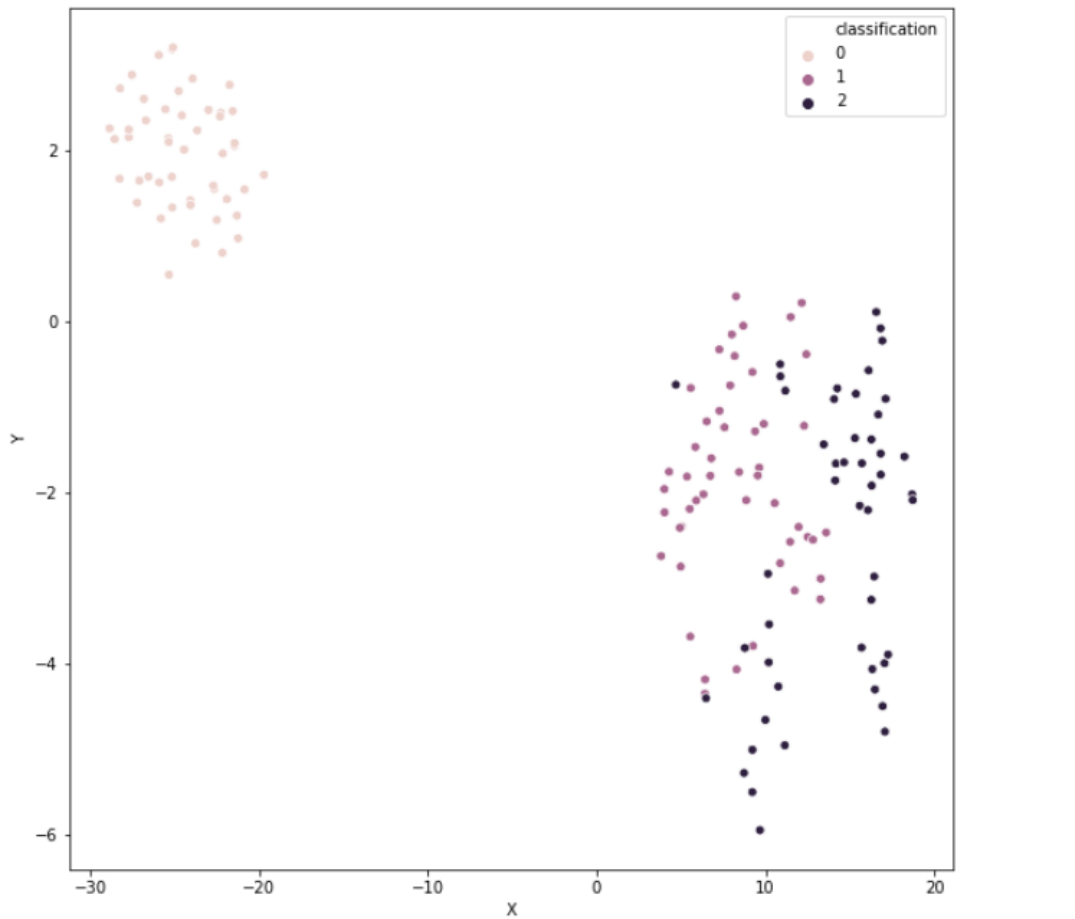
```
import seaborn as sns
tsne_df = pd.DataFrame({'X' : points[:,0],
                        'Y' : points[:,1],
                         'classification' : iris.target
                        })

tsne_df.head()
```

	X	Y	classification
0	-25.380981	2.141409	0
1	-21.572531	2.459061	0
2	-22.676508	1.549442	0
3	-21.920462	1.428031	0
4	-25.925230	1.625245	0

2. t-sne

❖ t-sne 실습

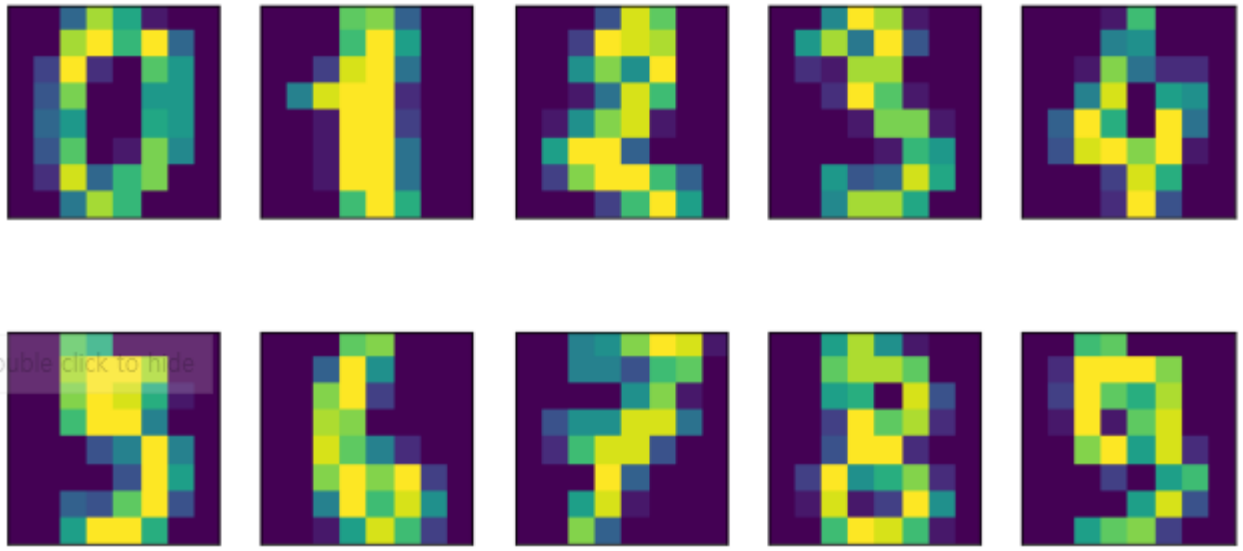
```
1 import seaborn as sns
2 plt.figure(figsize = (10,10))
3 sns.scatterplot(x = "X", y = 'Y', hue = 'classification', legend = 'full', data = tsne_df)
4 plt.show()
```



2. t-sne

❖ t-sne 실습

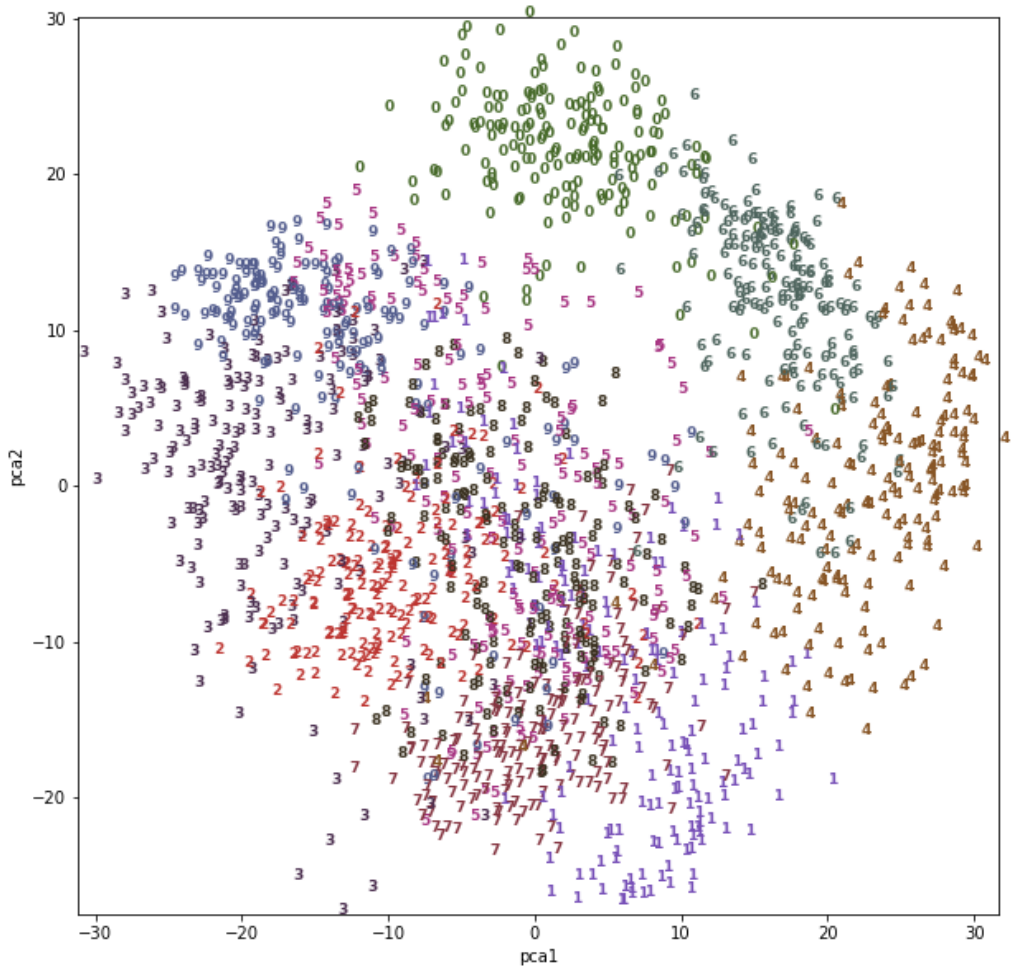
```
from sklearn.datasets import load_digits
digits = load_digits()

fig, axes = plt.subplots(2, 5, figsize=(10, 5),
                         subplot_kw={'xticks':(), 'yticks': ()})
for ax, img in zip(axes.ravel(), digits.images):
    ax.imshow(img)
```



2. t-sne

❖ t-sne 실습

```
# PCA 모델을 생성합니다
pca = PCA(n_components=2)
pca.fit(digits.data)
# 처음 두 개의 주성분으로 숫자 데이터를 변환합니다
digits_pca = pca.transform(digits.data)
colors = ["#476A2A", "#7851B8", "#BD3430", "#4A2D4E", "#875525",
          "#A83683", "#4E655E", "#853541", "#3A3120","#535D8E"]
plt.figure(figsize=(10, 10))
plt.xlim(digits_pca[:, 0].min(), digits_pca[:, 0].max())
plt.ylim(digits_pca[:, 1].min(), digits_pca[:, 1].max())
for i in range(len(digits.data)):
    # 숫자 텍스트를 이용해 산점도를 그립니다
    plt.text(digits_pca[i, 0], digits_pca[i, 1], str(digits.target[i]),
             color = colors[digits.target[i]],
             fontdict={'weight': 'bold', 'size': 9})
plt.xlabel("pca1")
plt.ylabel("pca2")
```

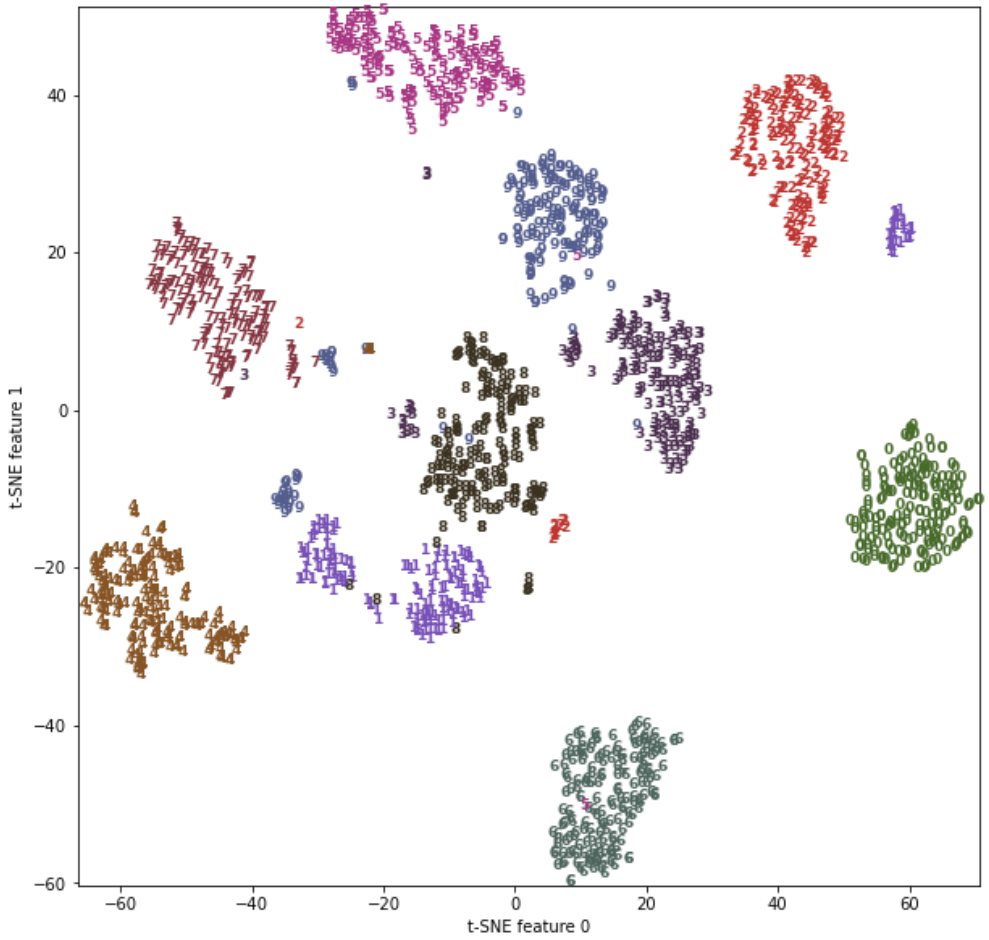


2. t-sne

❖ t-sne 실습

```
from sklearn.manifold import TSNE
tsne = TSNE(random_state=42)
# TSNE에는 transform 메소드가 없으므로 대신 fit_transform을 사용합니다
digits_tsne = tsne.fit_transform(digits.data)
```

```
plt.figure(figsize=(10, 10))
plt.xlim(digits_tsne[:, 0].min(), digits_tsne[:, 0].max() + 1)
plt.ylim(digits_tsne[:, 1].min(), digits_tsne[:, 1].max() + 1)
for i in range(len(digits.data)):
    # 숫자 텍스트를 이용해 산점도를 그립니다
    plt.text(digits_tsne[i, 0], digits_tsne[i, 1], str(digits.target[i]),
             color = colors[digits.target[i]],
             fontdict={'weight': 'bold', 'size': 9})
plt.xlabel("t-SNE feature 0")
plt.ylabel("t-SNE feature 1")
```



reference

- 모든 강의자료는 고려대 강필성 교수님 강의와 김성범 교수님 강의를 참고했음
- ratsgo’s blog ,https://ratsgo.github.io/
- 안드레아스 뮐러, 세라 가이도 지음, 박해선 옮김, "파이썬 라이브러리를 활용한 머신러닝", 한빛미디어(2017)
- https://bcho.tistory.com/1210
- https://datascienceschool.net/
- https://excelsior-cjh.tistory.com/167

감사합니다