UST 과학기술연합대학원대학교
UNIVERSITY OF SCIENCE & TECHNOLOGY

9강: 차원축소 PCA

인공지능 일반강좌 : 기계학습의 이해(L2-1)

Contents

01.
PCA 소개

이홍석 (hsyi@kisti.re.kr)

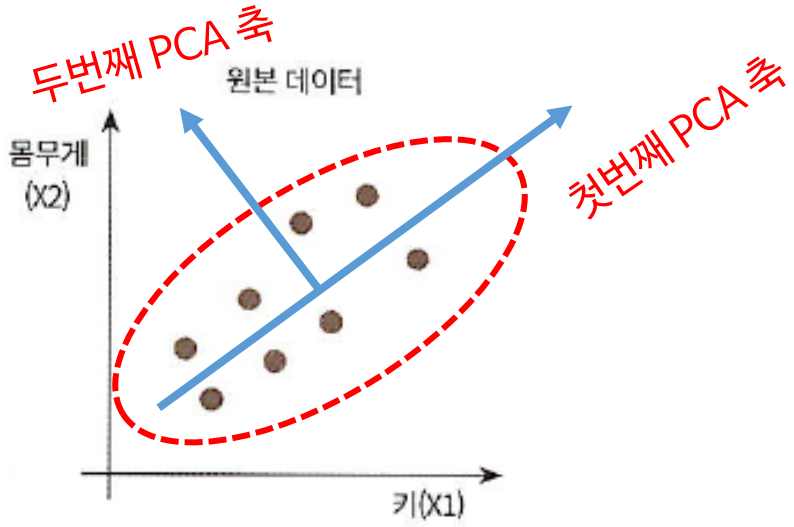
PCA 주성분분석(1)

- **차원축소는 다차원의 피처를 차원축소해 피처 개수를 줄임**
- **차원축소**
  - ✓ **피처 선택 (feature selection)**
    - 종속성이 강한 불필요한 피처는 제거하고 데이터의 특성을 잘 나타내는 주요 피처만을 선택
  - ✓ **피처 추출 (feature extraction)**
    - 기존 피처를 저차원의 중요 피처로 압축해서 추출, 완전히 다른 피처가 된다.

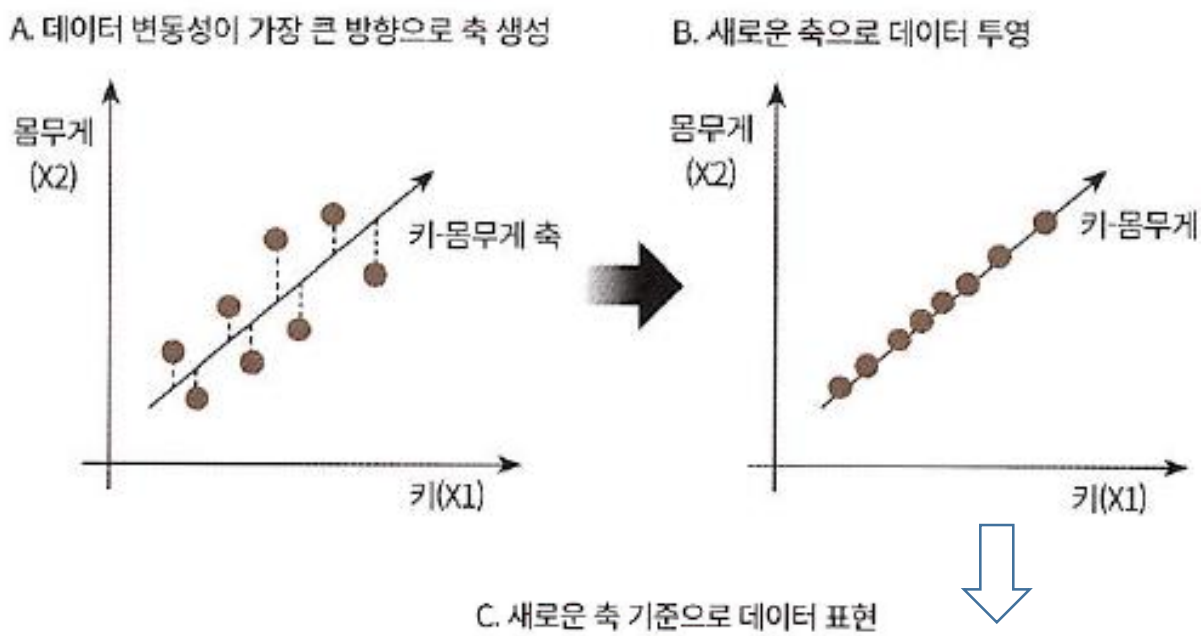
PCA 주성분분석(2)

- PCA 개요 (Principal Component Analysis)
  - ✓ 대표적인 차원축소 기법.
  - ✓ 가장 높은 분산을 가지는 데이터의 축을 찾아 이 축으로 차원을 축소하면,
  - ✓ 주성분. 분산이 데이터의 특성을 가장 잘 나타내는 것으로 간주

데이터의 변동성이 큰 방향은? 변동성(Variance)

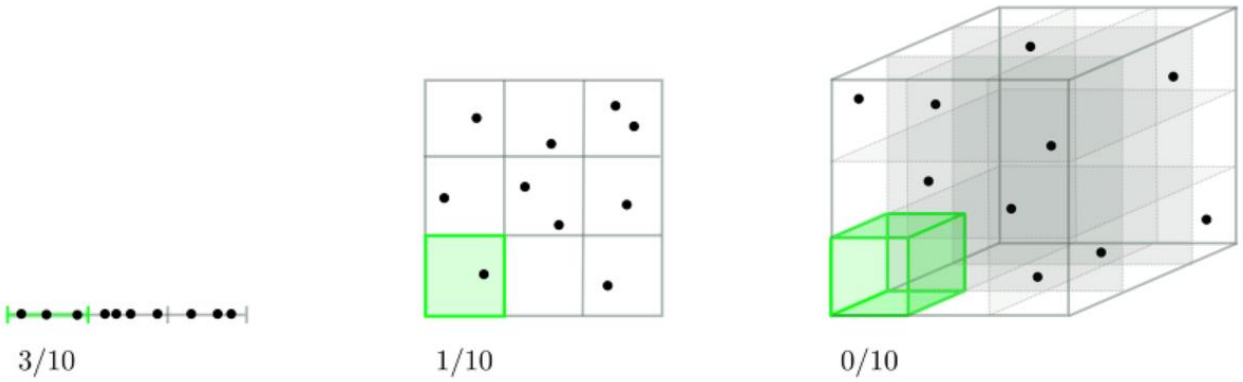


PCA 주성분분석(3)


A. 데이터 변동성이 가장 큰 방향으로 축 생성
몸무게
(X2)
키-몸무게 축
키(X1)
B. 새로운 축으로 데이터 투영
몸무게
(X2)
키-몸무게
키(X1)
C. 새로운 축 기준으로 데이터 표현
키-몸무게 축

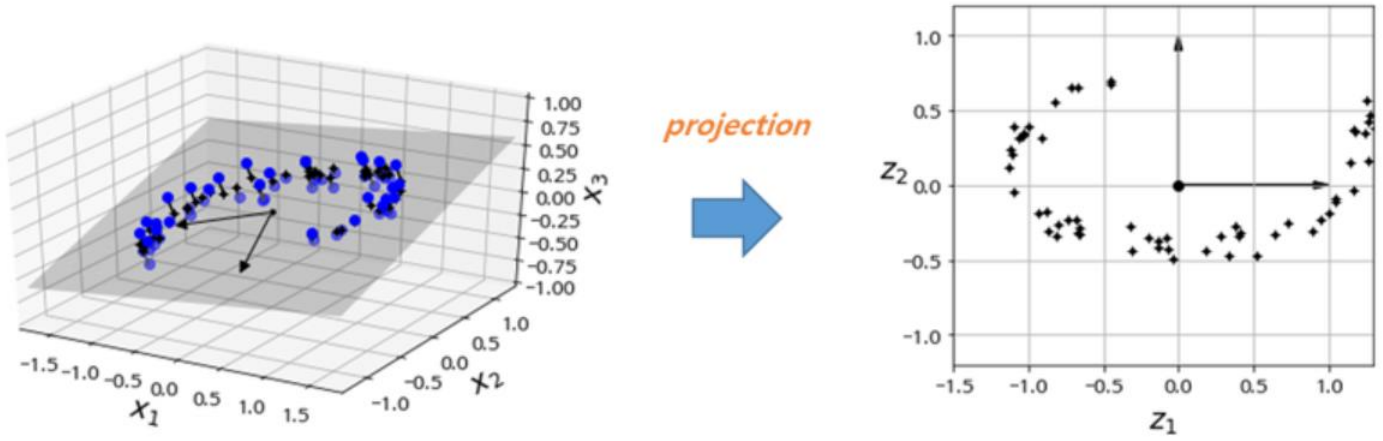
PCA 주성분분석(4)

- **차원의 저주**
  - ✓ **데이터 셋의 특성이 많아지면 각 특성인 하나의 차원(dimension) 증가한다.**
    - 데이터의 차원이 증가할 수록 데이터 공간의 부피가 기하 급수적으로 증가하기 때문에, 데이터의 밀도는 차원이 증가할 수록 희소(sparse)해진다.
  - ✓ **저주를 해결하기 위한 방법**
    - 데이터의 밀도가 높아질 때까지 학습 데이터 셋의 크기를 늘리는 것. 불가능
    - 이유는, 데이터셋의 크기에 비해 차원은 기하급수적으로 커지기 때문



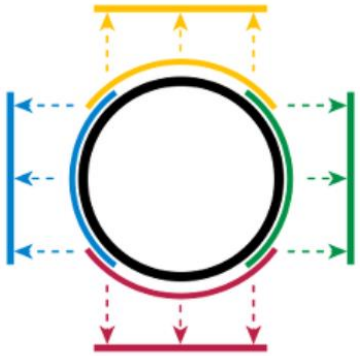
PCA 주성분분석(5)

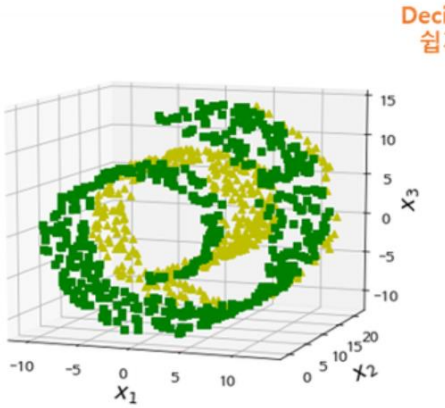
- **차원 축소를 위한 접근 방법**
  - ✓ **투영(Projection)**
    - 학습 데이터셋은 고차원 공간에서 저차원 부분 공간(subspace)에 위치
    - 고차원의 데이터의 특성 중 일부 특성으로 데이터를 표현할 수 있다
  - ✓ **(예시) 3차원 공간상의 데이터를 2차원 부분 공간으로 투영(porjection)시킴**

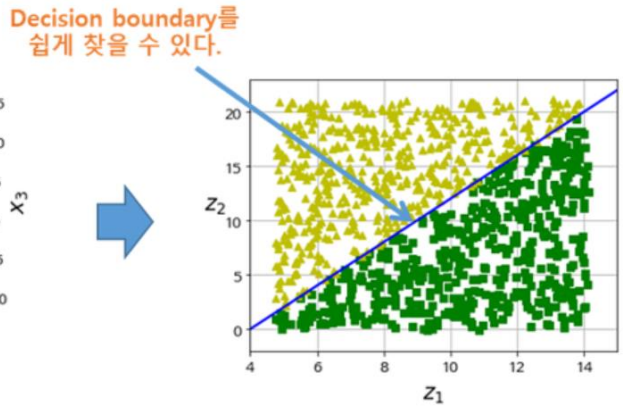


PCA 주성분분석(6)

- **차원 축소를 위한 접근 방법**
  - ✓ **매니폴드(manifold)**
    - 매니폴드는 다양체라고도 하며 국소적으로 유클리드 공간과 닮은 위상 공간이다
    - (예) 1차원 매니폴드,
    - (예) 2차원 스위스 롤(swiss roll) 데이터 셋이며, 2차원-매니폴드

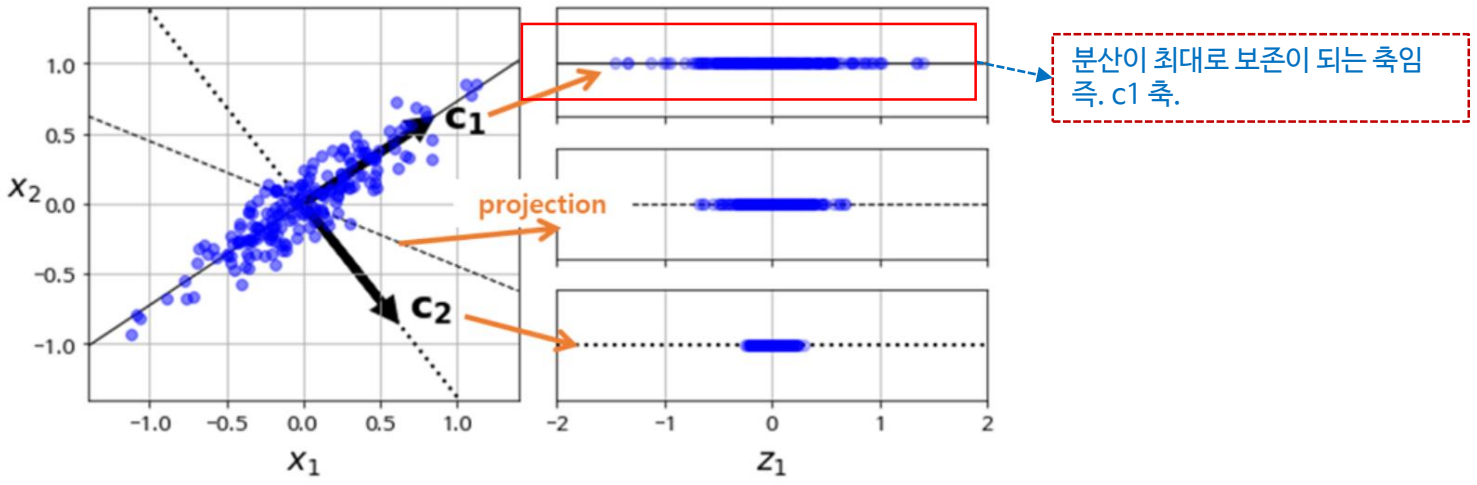






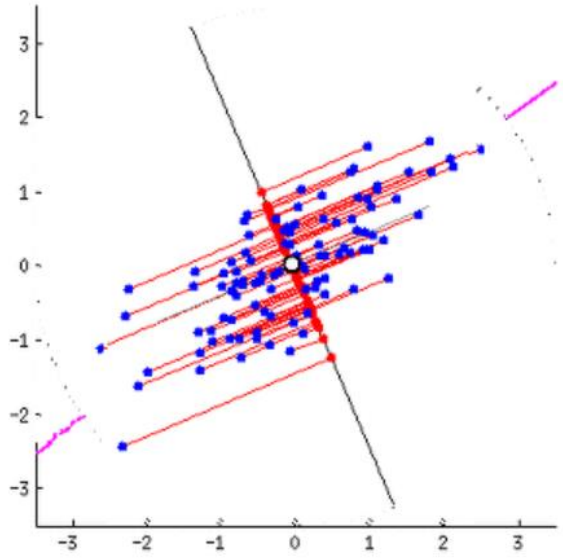
PCA 주성분분석(7)

- PCA (주성분 분석)
  - ✓ 데이터에 가장 가까운 초평면을 구한 다음, 데이터를 이 초평면에 투영 함.
- 분산 보존
  - ✓ 먼저 적절한 초평면을 선택해야 한다.
  - ✓ PCA는 데이터의 분산이 최대가 되는 축을 찾는다.
    - 원본 데이터셋과 투영된 데이터셋 간의 평균제곱거리를 최소화 하는 축



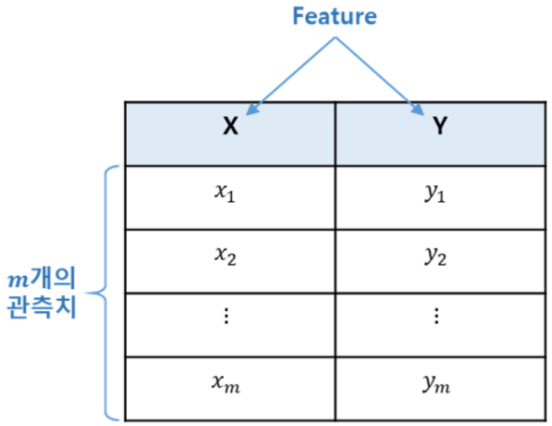
PCA 주성분분석(8)

- 주성분 (Principal Component)
  - ✓ 학습 데이터셋에서 분산이 최대인 축(axis)을 찾는다.
  - ✓ 이렇게 찾은 첫번째 축과 직교(orthogonal)하면서 분산이 최대인 두 번째 축을 찾는다.
  - ✓ 첫 번째 축과 두 번째 축에 직교하고 분산을 최대한 보존하는 세 번째 축을 찾는다.
  - ✓ 1~3과 같은 방법으로 데이터셋의 차원(특성 수)만큼의 축을 찾는다



PCA 주성분분석(9)

- 공분산(Covariance)
  - ✓ 공분산(covariance)은 2개의 특성(또는 변수)간의 상관 정도를 나타낸 값
  - ✓ (예) 두 개의 특성에 대해 공분산을 구하면



$$E[\mathbf{X}] = \mu, \quad E[\mathbf{Y}] = v$$

X, Y에 대한 기대값 (평균값)

공분사 cov(X,Y)

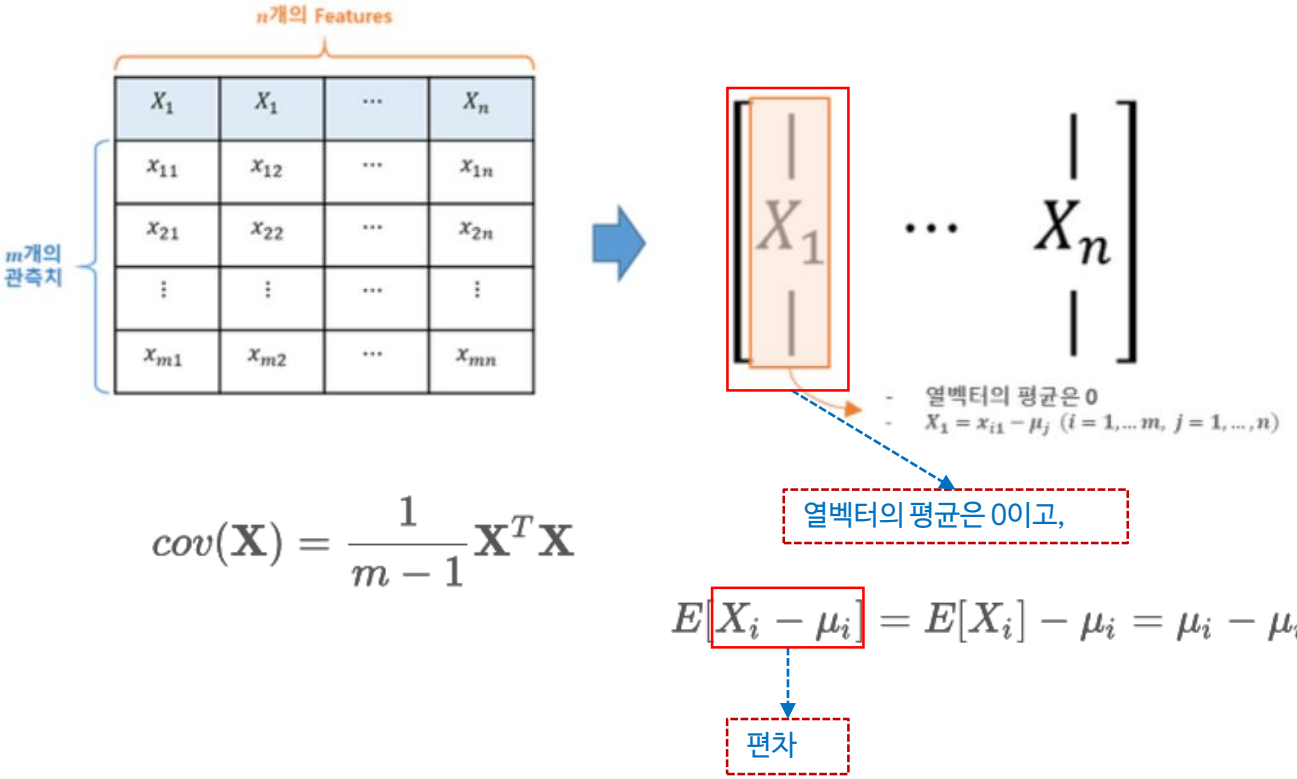
$$\begin{aligned} cov(\mathbf{X}, \mathbf{Y}) &= E[(\mathbf{X} - \mu)(\mathbf{Y} - v)] \\ &= E[\mathbf{XY} - v\mathbf{X} - \mu\mathbf{Y} + uv] \\ &= E[\mathbf{XY}] - vE[\mathbf{X}] - \mu E[\mathbf{Y}] + uv \\ &= E[\mathbf{XY}] - vu - uv + uv \\ &= E[\mathbf{XY}] - uv \end{aligned}$$

$$\begin{aligned} E[\mathbf{XY}] - uv &= \frac{1}{m-1}\sum_{i=1}^{m} X_i Y_i - uv \\ &= \frac{1}{m-1}\langle \mathbf{X}, \mathbf{Y} \rangle - uv \\ &= \frac{1}{m-1}\mathbf{X}^T\mathbf{Y} - uv \end{aligned}$$

공분산을 벡터로 표시

PCA 주성분분석(10)

- n개의 특성과 m개의 관측치로 구성된 데이터에 대한 공분산은?



$$cov(\mathbf{X}) = \frac{1}{m-1}\mathbf{X}^T\mathbf{X}$$

PCA 주성분분석(11)

C는 공분산, 벡터 e는 고유벡터(eigenvector), 고유벡터의 열벡터를 주성분 이라고 한다.

PCA는 $Var\left[\mathbf{X}\vec{e}\right] = \vec{e}^{T}\mathbf{C}\vec{e}$ 를 목적함수로 하는 최대화 문제이며 이때 제약조건은 $\|\vec{e}\|^{2} = 1$ 이다.

$\mathbf{C}\vec{e} = \lambda\vec{e}$ 를 만족하는 $\vec{e}$가 바로 분산 $Var\left[\mathbf{X}\vec{e}\right]$ 를 최대화한다.

고유치(eigenvalue)

$$
\begin{aligned}
Var\left[\mathbf{X}\vec{e}\right] &= \frac{1}{m-1}\sum_{i=1}^{m}\left[X\vec{e} - E\left(X\vec{e}\right)\right]^{2} \\
&= \frac{1}{m-1}\sum_{i=1}^{m}\left[X\vec{e} - E(X)\vec{e}\right]^{2}, \quad (E(X) = 0) \\
&= \frac{1}{m-1}\sum_{i=1}^{m}\left(X\vec{e}\right)^{2} \\
&= \frac{1}{m-1}\left(\mathbf{X}\vec{e}\right)^{T}\left(\mathbf{X}\vec{e}\right) \\
&= \frac{1}{m-1}\vec{e}^{T}\mathbf{X}^{T}\mathbf{X}\vec{e} \\
&= \vec{e}^{T}\left(\frac{\mathbf{X}^{T}\mathbf{X}}{m-1}\right)\vec{e}, \quad \left(\frac{\mathbf{X}^{T}\mathbf{X}}{m-1} = \mathbf{C}\right) \\
&= \vec{e}^{T}\mathbf{C}\vec{e}
\end{aligned}
$$

라그랑지안 함수

$$L\left(\vec{e},\lambda\right) = \vec{e}^{T}\mathbf{C}\vec{e} - \lambda\left(\vec{e}^{T}\vec{e} - 1\right)$$

$$
\begin{aligned}
\frac{\partial L}{\partial \vec{e}} &= \left(\mathbf{C} + \mathbf{C}^{T}\right)\vec{e} - 2\lambda\vec{e} \\
&= 2\mathbf{C}\vec{e} - 2\lambda\vec{e} = 0
\end{aligned}
$$

$$\therefore \mathbf{C}\vec{e} = \lambda\vec{e}$$

$$\therefore \mathbf{C} = \vec{e}\lambda\vec{e}^{T}$$

PCA 주성분분석(12)

- **사이킷런에서 PCA는**
  - ✓ **고유값 분해가 아니고, 특이값 분해(SVD)이용한다.**
  - ✓ **PCA 계산에서 SVD를 사용하는 이유는 무엇일까?**
    - 공분산 행렬을 메모리상에 가지고 있어야하는 반면, SVD는 공분산 행렬을 따로 메모리에 저장할 필요가 없으므로 효율적이기 때문이다(?)
  - ✓ **데이터 셋 X에 대한 SVD는 다음과 같다.**

$$\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$$

- $\mathbf{U}$: $m \times m$ 직교행렬 → $\mathbf{X}\mathbf{X}^T$ 의 eigenvector, $\mathbf{X}$의 **left singular vector**
- $\mathbf{V}$: $n \times n$ 직교행렬 → $\mathbf{X}^T\mathbf{X}$ 의 eigenvector, $\mathbf{X}$의 **right singular vector**라 하며, PCA의 주성분행렬
- Σ : $m \times n$ 대각행렬 → $\mathbf{X}\mathbf{X}^T$ 또는 $\mathbf{X}^T\mathbf{X}$ 의 **eigenvalue의 제곱근**을 대각원소로 하는 행렬

02. 따라해보기
PCA 실습

이홍석 (hsyi@kisti.re.kr)

PCA 적용 –기본 연습(1)

```
import pandas as pd
```

```
# Eating, exercise habbit and their body shape

df = pd.DataFrame(columns=['calory', 'breakfast', 'lunch', 'dinner', 'exercise', 'body_shape'])
```

```
df.loc[0] = [1200, 1, 0, 0, 2, 'Skinny']
df.loc[1] = [2800, 1, 1, 1, 1, 'Normal']
df.loc[2] = [3500, 2, 2, 1, 0, 'Fat']
df.loc[3] = [1400, 0, 1, 0, 3, 'Skinny']
df.loc[4] = [5000, 2, 2, 2, 0, 'Fat']
df.loc[5] = [1300, 0, 0, 1, 2, 'Skinny']
df.loc[6] = [3000, 1, 0, 1, 1, 'Normal']
df.loc[7] = [4000, 2, 2, 2, 0, 'Fat']
df.loc[8] = [2600, 0, 2, 0, 0, 'Normal']
df.loc[9] = [3000, 1, 2, 1, 1, 'Fat']
```

PCA 적용 –기본 연습(2)

```
df.head(10)
```

	calory	breakfast	lunch	dinner	exercise	body_shape
0	1200	1	0	0	2	Skinny
1	2800	1	1	1	1	Normal
2	3500	2	2	1	0	Fat
3	1400	0	1	0	3	Skinny
4	5000	2	2	2	0	Fat
5	1300	0	0	1	2	Skinny
6	3000	1	0	1	1	Normal
7	4000	2	2	2	0	Fat
8	2600	0	2	0	0	Normal
9	3000	1	2	1	1	Fat

PCA 적용 –기본 연습(3)

```
# X is feature vectors
X = df[['calory', 'breakfast', 'lunch', 'dinner', 'exercise']]
```

	calory	breakfast	lunch	dinner	exercise
0	1200	1	0	0	2
1	2800	1	1	1	1
2	3500	2	2	1	0
3	1400	0	1	0	3
4	5000	2	2	2	0
5	1300	0	0	1	2
6	3000	1	0	1	1
7	4000	2	2	2	0
8	2600	0	2	0	0

PCA 적용 –기본 연습(4)

```
Y = df[['body_shape']]; Y.head(10)
```

	body_shape
0	Skinny
1	Normal
2	Fat
3	Skinny
4	Fat
5	Skinny
6	Normal
7	Fat
8	Normal
9	Fat

PCA 적용 -기본 연습(5)

3. rescaling feature vectors to all have the same scale

```
from sklearn.preprocessing import StandardScaler
x_std = StandardScaler().fit_transform(X)
```

```
x_std
```

```
11]: array([[-1.35205803,  0.        , -1.3764944 , -1.28571429,  1.        ],
       [ 0.01711466,  0.        , -0.22941573,  0.14285714,  0.        ],
       [ 0.61612771,  1.29099445,  0.91766294,  0.14285714, -1.        ],
       [-1.18091145, -1.29099445, -0.22941573, -1.28571429,  2.        ],
       [ 1.89972711,  1.29099445,  0.91766294,  1.57142857, -1.        ],
       [-1.26648474, -1.29099445, -1.3764944 ,  0.14285714,  1.        ],
       [ 0.18826125,  0.        , -1.3764944 ,  0.14285714,  0.        ],
       [ 1.04399418,  1.29099445,  0.91766294,  1.57142857, -1.        ],
       [-0.15403193, -1.29099445,  0.91766294, -1.28571429, -1.        ],
       [ 0.18826125,  0.        ,  0.91766294,  0.14285714,  0.        ]])
```

PCA 적용 -기본 연습(6)

4. Covariance Matrix of features

```
import numpy as np
```

```
# features are columns from x_std
features = x_std.T
covariance_matrix = np.cov(features)
print(covariance_matrix)
```

```
[[ 1.11111111  0.88379717  0.76782385  0.89376551 -0.93179808]
 [ 0.88379717  1.11111111  0.49362406  0.81967902 -0.71721914]
 [ 0.76782385  0.49362406  1.11111111  0.40056715 -0.76471911]
 [ 0.89376551  0.81967902  0.40056715  1.11111111 -0.63492063]
 [-0.93179808 -0.71721914 -0.76471911 -0.63492063  1.11111111]]
```

PCA 적용 –기본 연습(7)

5. 고유벡터, 고유값, 공분산

```
eig_vals, eig_vecs = np.linalg.eig(covariance_matrix)
print('Eigenvectors ₩n%s' %eig_vecs)
```

```
Eigenvectors
[[ 0.508005    0.0169937  -0.84711404  0.11637853  0.10244985]
 [ 0.44660335  0.36890361  0.12808055 -0.63112016 -0.49973822]
 [ 0.38377913 -0.70804084  0.20681005 -0.40305226  0.38232213]
 [ 0.42845209  0.53194699  0.3694462   0.22228235  0.58954327]
 [-0.46002038  0.2816592  -0.29450345 -0.61341895  0.49601841]]
```

PCA 적용 –기본 연습(8)

```
print('₩nEigenvalues ₩n%s' %eig_vals)
```

```
Eigenvalues
[4.0657343  0.8387565  0.07629538 0.27758568 0.2971837 ]
```

```
# We reduce dimension to 1 dimension, since 1 eigenvector has 73% (enough) variances
eig_vals[0] / sum(eig_vals)
```

```
0.7318321731427544
```

PCA 적용 –기본 연습(9)

6. Project data point onto selected Eigen Vector

```
projected_X = x_std.dot(eig_vecs.T[0])
```

```
projected_X
```

19]:
```
array([-2.22600943, -0.0181432 ,  1.76296611, -2.73542407,  3.02711544,
       -2.14702579, -0.37142473,  2.59239883, -0.39347815,  0.50902498])
```

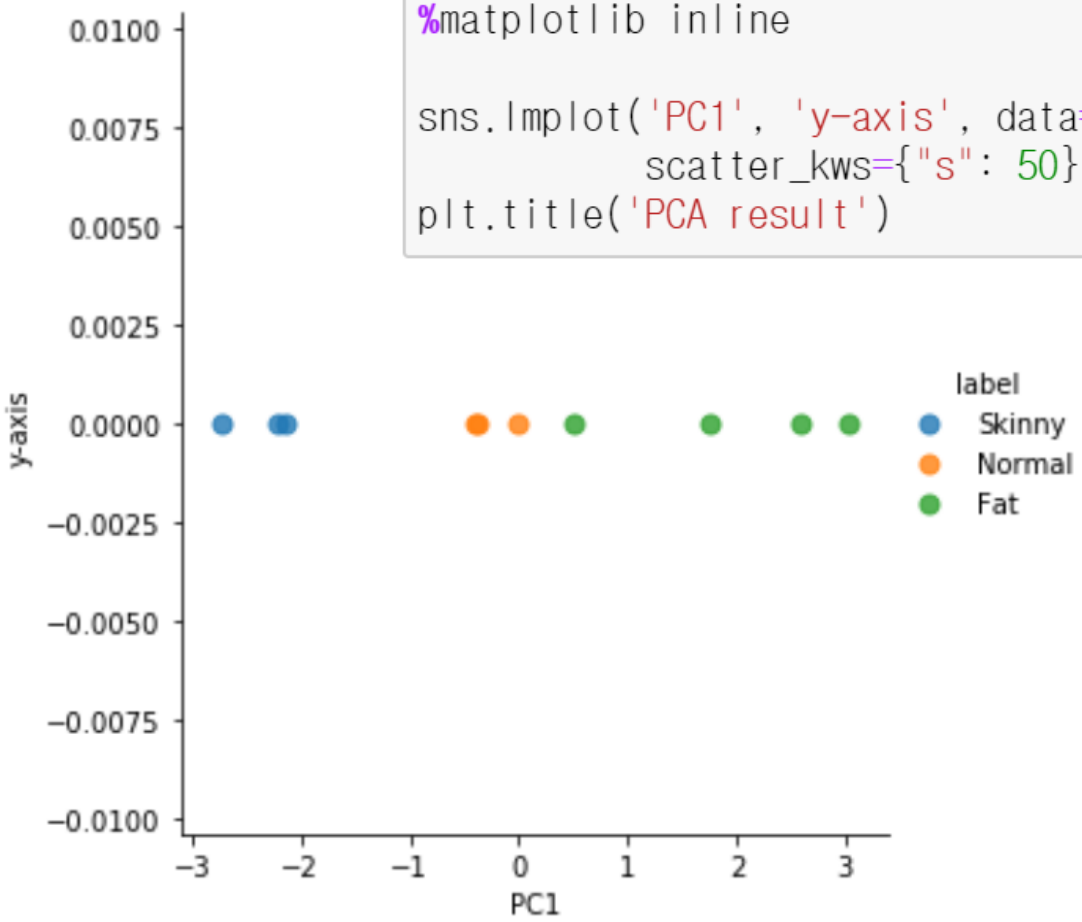
```
result = pd.DataFrame(projected_X, columns=['PC1'])
result['y-axis'] = 0.0
result['label'] = Y
```

PCA 적용 –기본 연습(10)

	PC1	y-axis	label
0	-2.226009	0.0	Skinny
1	-0.018143	0.0	Normal
2	1.762966	0.0	Fat
3	-2.735424	0.0	Skinny
4	3.027115	0.0	Fat
5	-2.147026	0.0	Skinny
6	-0.371425	0.0	Normal
7	2.592399	0.0	Fat
8	-0.393478	0.0	Normal
9	0.509025	0.0	Fat

PCA 적용 –기본 연습(11)

```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

sns.lmplot('PC1', 'y-axis', data=result, fit_reg=False,
           scatter_kws={"s": 50}, hue="label")
plt.title('PCA result')
```



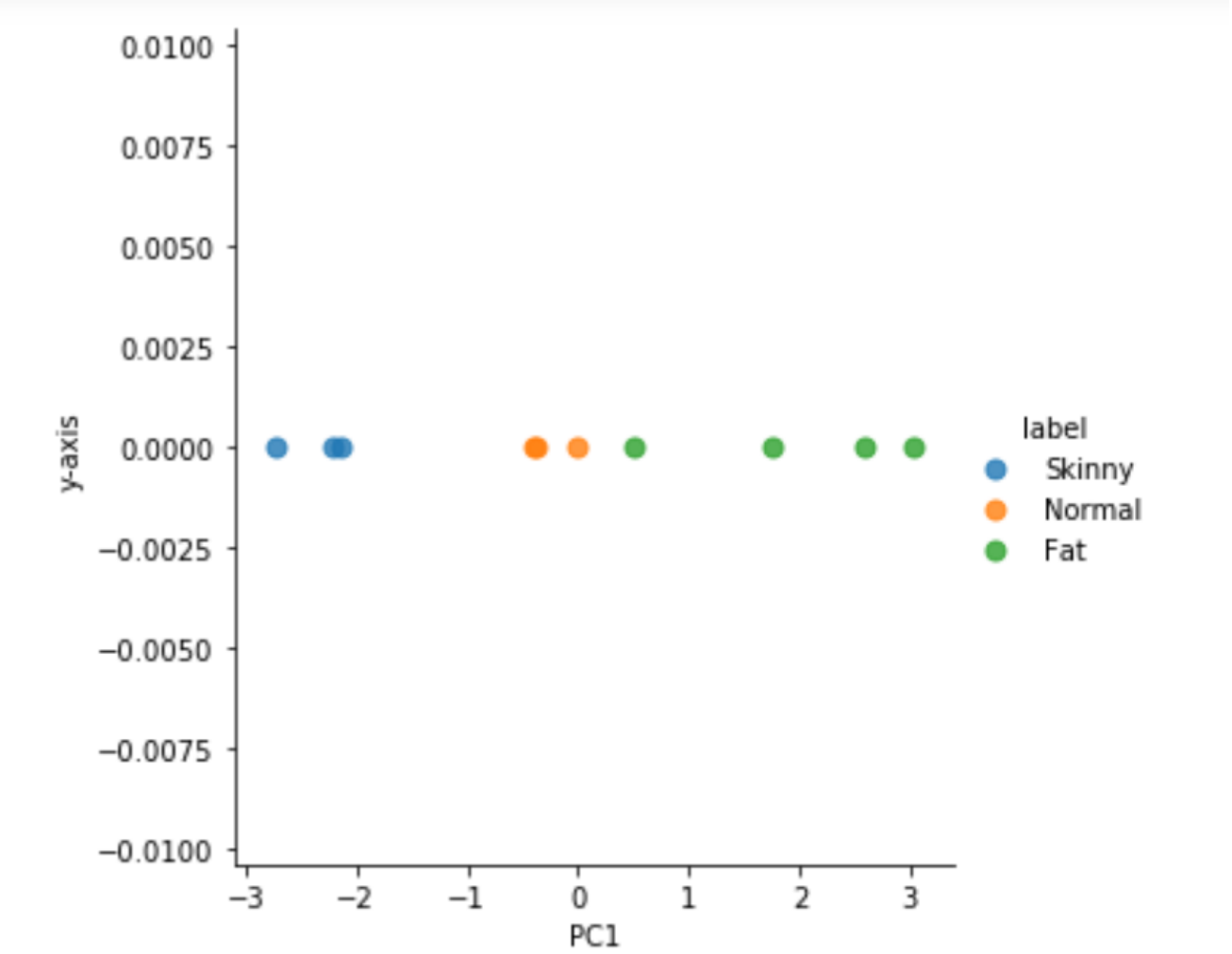
PCA 적용 –기본 연습(12)

bonus - scikit-learn PCA

```
from sklearn import decomposition
pca = decomposition.PCA(n_components=1)
sklearn_pca_x = pca.fit_transform(x_std)
```

```
sklearn_result = pd.DataFrame(sklearn_pca_x, columns=['PC1'])
sklearn_result['y-axis'] = 0.0
sklearn_result['label'] = Y

sns.lmplot('PC1', 'y-axis', data=sklearn_result, fit_reg=False,
           scatter_kws={"s": 50}, # marker size
           hue="label") # color
```

PCA 적용 –기본 연습(13)



03. PCA 적용
붓꽃 데이터

이홍석 (hsyi@kisti.re.kr)

PCA 적용 – 붓꽃 (1)

붓꽃 4개의 데이터 셋트를 2개의 차원으로 축소해라.

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

```
iris = load_iris()
columns = ['sepal_length','sepal_width','petal_length','petal_width']
irisDF = pd.DataFrame(iris.data , columns=columns)
irisDF['target']=iris.target
irisDF.head(2)
```

붓꽃 데이터 로드 API

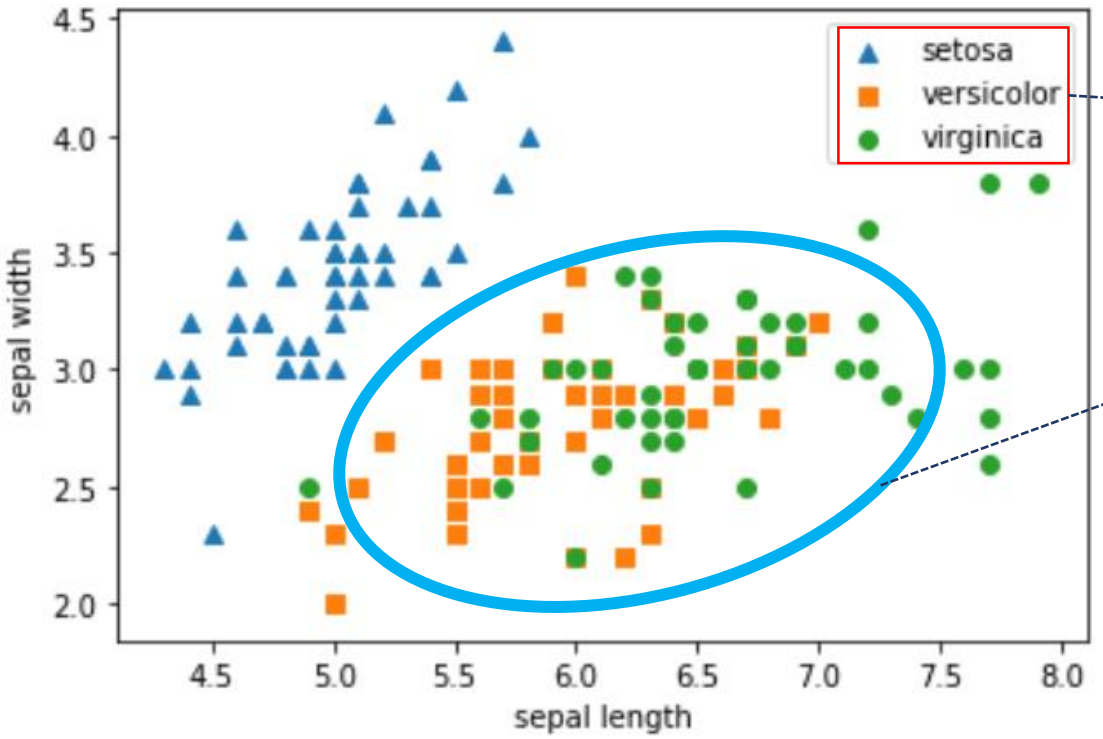
	sepal_length	sepal_width	petal_length	petal_width	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0

꽃잎 2개를 먼저 그림

PCA 적용 – 붓꽃 (2)

```
markers=['^', 's', 'o']
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])
```

세모, 네모, 동그라미



0,
1,
2

Versicolor와
virginica
2개를 분류가
쉽지 않음

PCA 적용 – 붓꽃 (3)

개별 속성을 표준화 스케일링. 표준 정규 분표로 평균 0, 분산 1임.

```
from sklearn.preprocessing import StandardScaler

iris_scaled = StandardScaler().fit_transform(irisDF)
```

```
from sklearn.decomposition import PCA

pca = PCA(n_components=2)

pca.fit(iris_scaled)
iris_pca = pca.transform(iris_scaled)
print(iris_pca.shape)
```

PCA로 변환할 차원의 개수로 2차원.

```
(150, 2)
```

새로운 Iris_pca 객체 변수 (150,2)개로 변환.

PCA 적용 – 붓꽃 (4)

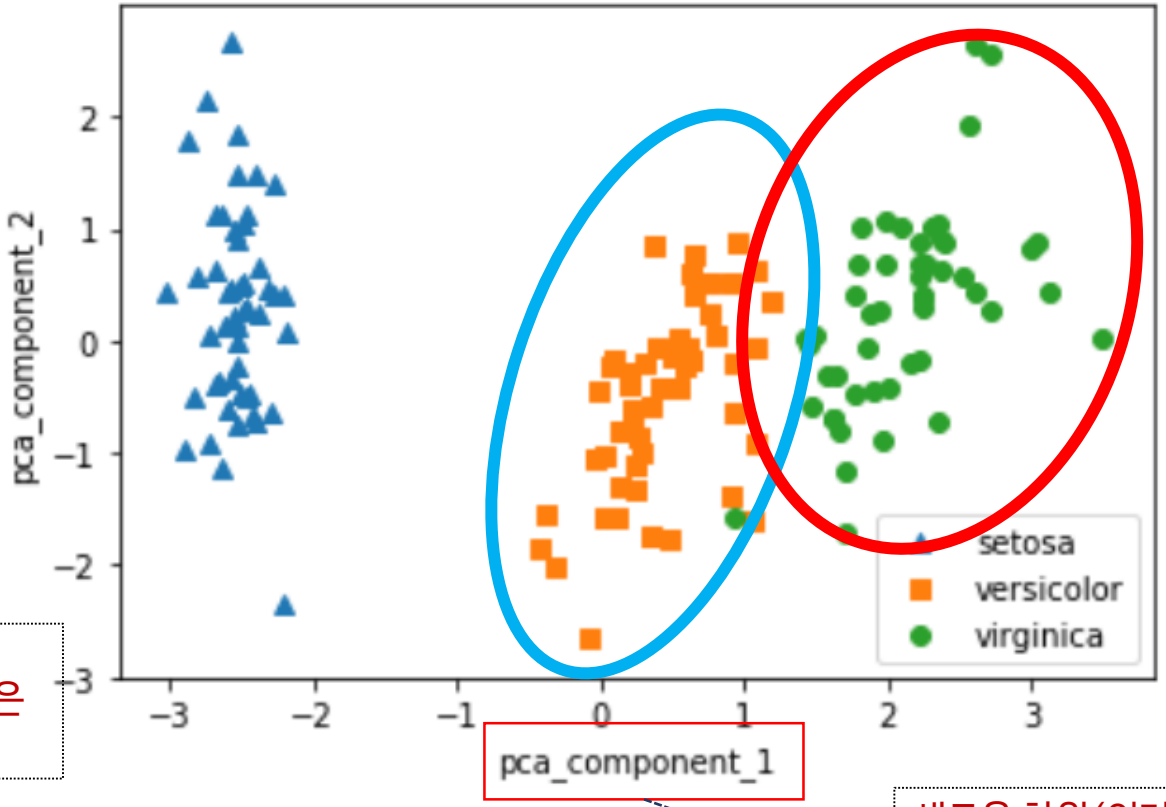
```
pca_columns=['pca_component_1','pca_component_2']
irisDF_pca = pd.DataFrame(iris_pca,columns=pca_columns)
irisDF_pca['target']=iris.target
irisDF_pca.head(3)
```

새로운 데이터 컬럼 이름으로
pca_component_1 등 사용함

	pca_component_1	pca_component_2	target
0	-2.576120	0.474499	0
1	-2.415322	-0.678092	0
2	-2.659333	-0.348282	0

```
markers=['^', 's', 'o']
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()
```

PCA 적용 – 붓꽃 (5)



2개의
Pca 요소 합은
95%

```
print(pca.explained_variance_ratio_)
[0.76740358 0.18282727]
```

새로운 차원(의미는 무엇인가?)
원본 데이터의 변동성(76.7%)로
매우
잘 반영되었다.

PCA 적용 – 붓꽃 (6)

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score

rcf = RandomForestClassifier(random_state=156)

scores = cross_val_score(rcf, iris.data, iris.target,scoring='accuracy',cv=3)

print(scores)
```

렘덤포레스트 분류기

3개의 교차 검증기 사용하여 cross_val_score를 계산

```
[0.98 0.94 0.96]
```

```
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(scores_pca)
```

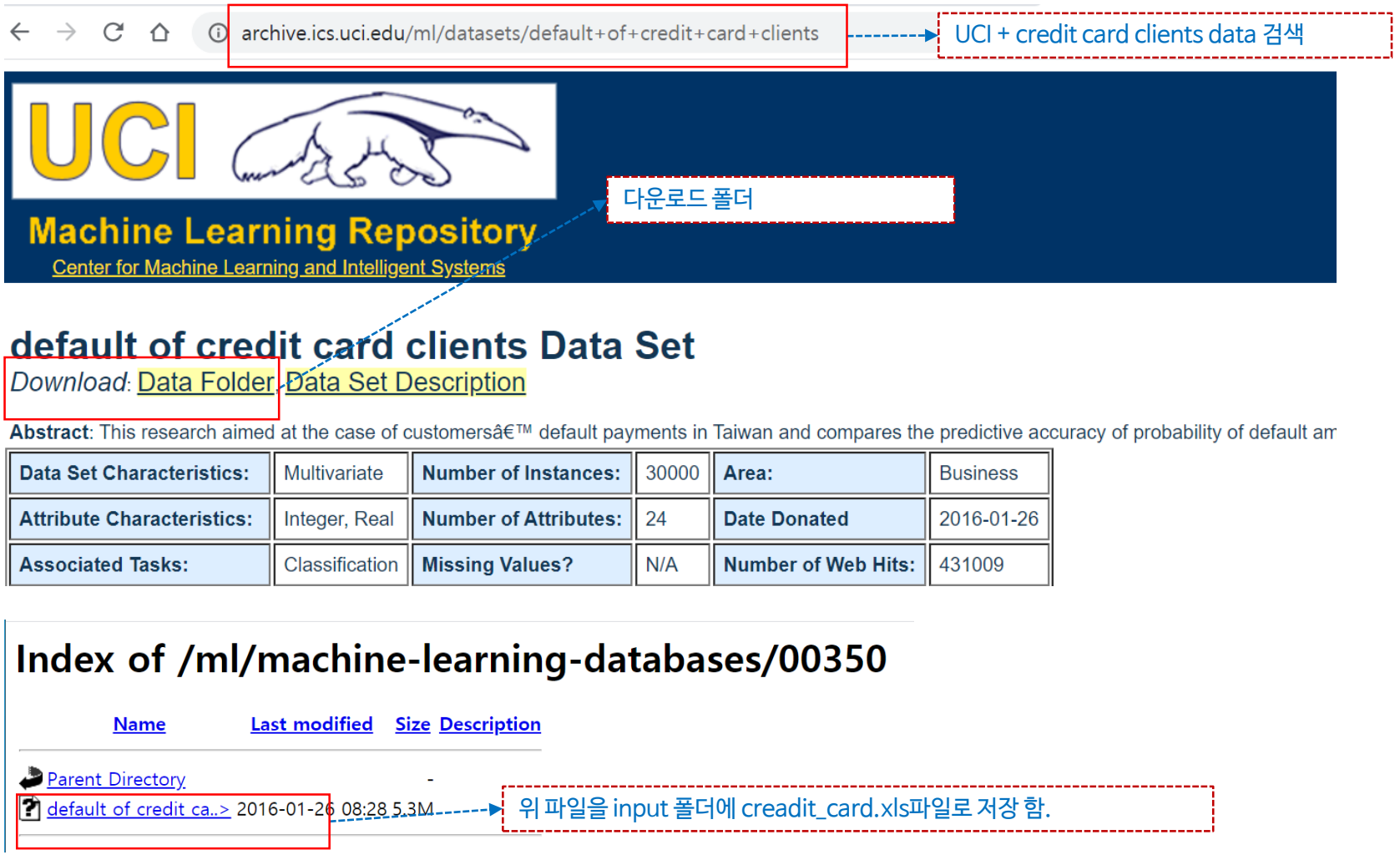
```
[0.98 0.98 1.  ]
```

원본 예측 정확도 보다 PCA가 좀 더 정확하다. 단 붓꽃의 경우. 일반적으로 PCA가 원본 보다 예측정확도가 우수하다고 말하기 어렵다. 붓꽃은 PCA로 충분히 분류되는 예제일 뿐.

04. PCA 적용
신용카드 고객 데이터

이홍석 (hsyi@kisti.re.kr)

PCA 적용 – 신용카드 고객 데이터(1)



PCA 적용 – 신용카드 고객 데이터(2)

PCA 적용 - credit card 데이터 세트

In [1]:

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

#df = pd.read_excel('./input/credit_card.xls', sheet_name='Data')
df = pd.read_excel('./input/credit_card.xls')

print(df.shape);

df.head(5)
```

```
(30000, 25)
```

Out[1]:

판다스에는 엑셀 파일을 읽을 수 있음. Read_excel()

시트 명은 생략 할 수 있음. 엑셀에서 시트가 1개 일때.

신용카드 데이터 세트는 30,000개 레코드 열(row), 25개 속서 컬럼을 가지고 있음.

PCA 적용 – 신용카드 고객 데이터(3)

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	...
0	1	20000	2	2	1	24	2	2	-1	-1	...
1	2	120000	2	2	2	26	-1	2	0	0	...
2	3	90000	2	2	2	34	0	0	0	0	...
3	4	50000	2	2	1	37	0	0	0	0	...
4	5	50000	1	2	1	57	-1	0	-1	0	...

PAY_0을 PAY_1로 변경해서 순서를 마춤

5 rows × 25 columns

Target (라벨): 다음달 연체 여부로, 연체면 1, 정상납부가 0임.
Target이 너무 길어서 이름을 'default'로 변경할 것

BILL_AMT4	BILL_AMT5	BILL_AMT6	PAY_AMT1	PAY_AMT2	PAY_AMT3	PAY_AMT4	PAY_AMT5	PAY_AMT6	default payment next month
0	0	0	0	689	0	0	0	0	1
3272	3455	3261	0	1000	1000	1000	0	2000	1
14331	14948	15549	1518	1500	1000	1000	1000	5000	0
28314	28959	29547	2000	2019	1200	1100	1069	1000	0
20940	19146	19131	2000	36681	10000	9000	689	679	0

PCA 적용 – 신용카드 고객 데이터(4)

판다스 rename은 PAY_0을 PAY_1로 바꿈!

```
df.rename(columns={'PAY_0':'PAY_1','default payment next month':'default'}, inplace=True)
```

```
y_target = df['default']

X_features = df.drop('default', axis=1)
```

y_target는 다음달 연체 여부로 설정함
X_features는 default를 제외한 모든 것

```
df.keys()
```

```
Index(['ID', 'LIMIT_BAL', 'SEX', 'EDUCATION', 'MARRIAGE', 'AGE', 'PAY_1',
       'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6', 'BILL_AMT1', 'BILL_AMT2',
       'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1',
       'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6', 'default'],
      dtype='object')
```

PCA 적용 – 신용카드 고객 데이터(5)

```
X_features.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30000 entries, 0 to 29999
Data columns (total 24 columns):
ID              30000 non-null int64
LIMIT_BAL       30000 non-null int64
SEX             30000 non-null int64
EDUCATION       30000 non-null int64
MARRIAGE        30000 non-null int64
AGE             30000 non-null int64
PAY_1           30000 non-null int64
PAY_2           30000 non-null int64
PAY_3           30000 non-null int64
PAY_4           30000 non-null int64
PAY_5           30000 non-null int64
PAY_6           30000 non-null int64
BILL_AMT1       30000 non-null int64
BILL_AMT2       30000 non-null int64
BILL_AMT3       30000 non-null int64
BILL_AMT4       30000 non-null int64
BILL_AMT5       30000 non-null int64
BILL_AMT6       30000 non-null int64
PAY_AMT1        30000 non-null int64
PAY_AMT2        30000 non-null int64
PAY_AMT3        30000 non-null int64
PAY_AMT4        30000 non-null int64
PAY_AMT5        30000 non-null int64
PAY_AMT6        30000 non-null int64
dtypes: int64(24)
memory usage: 5.5 MB
```

PCA 적용 – 신용카드 고객 데이터(6)

pandas.DataFrame.corr

`DataFrame.corr(self, method='pearson', min_periods=1)`

Compute pairwise correlation of columns, excluding NA/null values.

method : *{'pearson', 'kendall', 'spearman'} or callable*

- pearson : standard correlation coefficient
- kendall : Kendall Tau correlation coefficient
- spearman : Spearman rank correlation
- callable: callable with input two 1d ndarrays

피어슨(pearson) 표준 상관관계지수

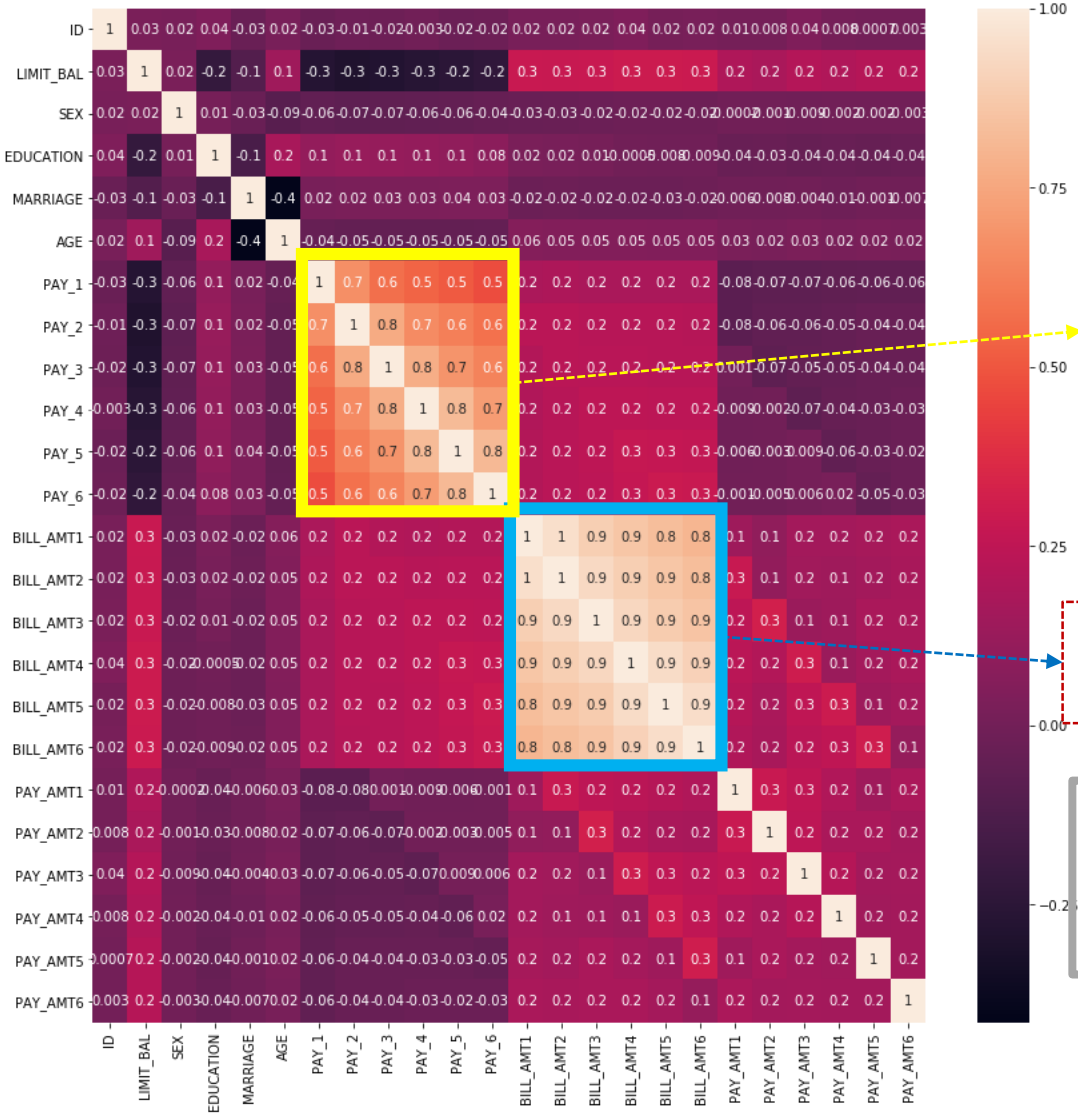
```
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

corr = X_features.corr()

plt.figure(figsize=(14,14))

sns.heatmap(corr, annot=True, fmt='.1g')
```

판다스에서 제공하는 각 속성들 사이의 상관관계도

seaborn.heatmap

`seaborn.heatmap(data, vmin=None, vmax=None, cmap=None, center=None, robust=False, annot=None, fmt='.2g', annot_kws=None,`

PCA 적용 – 신용카드 고객 데이터(7)



PAY1~PAY6까지의 속성도
상관관계가 높다.

BILL_ATM1~BILL_ATM6
6개 속성끼리 상관도가
대부분 0.9 이상으로 매우 높음!

6개의 속성을 2개의
축으로 PCA 해보면
어떨까?

PCA 적용 – 신용카드 고객 데이터(8)

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

#BILL_AMT1 ~ BILL_AMT6 까지 6개의 속성명 생성
cols_bill = ['BILL_AMT'+str(i) for i in range(1,7)]
print('대상 속성명:',cols_bill)

# 2개의 PCA 속성을 가진 PCA 객체 생성하고, explained_variance_ratio_ 계산 위해 fit( ) 호출
scaler = StandardScaler()
df_cols_scaled = scaler.fit_transform(X_features[cols_bill])
pca = PCA(n_components=2)
pca.fit(df_cols_scaled)

print('PCA Component별 변동성:', pca.explained_variance_ratio_)
```

```
대상 속성명: ['BILL_AMT1', 'BILL_AMT2', 'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6']
PCA Component별 변동성: [0.90555253 0.0509867 ]
```

단 2개의 PCA 컴포넌트 만으로도 6개의 속성의 변동성을 약 95%(0.90+0.05) 이상을 설명할 수 있다. 특히 주성분 축으로 90%의 변동성을 수용할 수 있을 정도로 6개 속성이 상관도가 매우 높다.

PCA 적용 – 신용카드 고객 데이터(9)

랜덤 포레스트 예측

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

rcf = RandomForestClassifier(n_estimators=300, random_state=156)

scores = cross_val_score(rcf, X_features, y_target, scoring='accuracy', cv=3 )

print('CV=3 인 경우의 개별 Fold세트별 정확도:',scores)
print('평균 정확도:{0:.4f}'.format(np.mean(scores)))
```

```
CV=3 인 경우의 개별 Fold세트별 정확도: [0.257  0.8209 0.784 ]
평균 정확도:0.6206
```

3개의 교차 검증 세트에서 평균 정확도는 62.1%

PCA 적용 – 신용카드 고객데이터(10)

PCA 예측

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# 원본 데이터셋에 먼저 StandardScaler적용
scaler = StandardScaler()
df_scaled = scaler.fit_transform(X_features)

# 6개의 Component를 가진 PCA 변환을 수행하고 cross_val_score( )로 분류 예측 수행.
pca = PCA(n_components=6)
df_pca = pca.fit_transform(df_scaled)
scores_pca = cross_val_score(rcf, df_pca, y_target, scoring='accuracy', cv=3)

print('CV=3 인 경우의 PCA 변환된 개별 Fold세트별 정확도:',scores_pca)
print('PCA 변환 데이터 셋 평균 정확도:{0:.4f}'.format(np.mean(scores_pca)))
```

```
CV=3 인 경우의 PCA 변환된 개별 Fold세트별 정확도: [0.775  0.7986 0.8034]
PCA 변환 데이터 셋 평균 정확도:0.7923
```

정확도 79.2%
PCA는 컴퓨터 비전에 많이 사용되고 있음.

Thank You!

www.ust.ac.kr