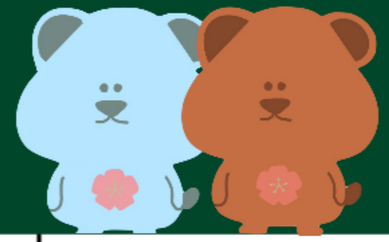


6.1 차원 축소 개요, 6.2 PCA

2팀 문원정

EWHA EURON

#01 차원 축소 개요

차원 축소란? 매우 많은 피처로 구성된 다차원 데이터 세트의 차원을 축소해 새로운 차원의 데이터 세트를 생성하는 것

일반적으로 차원이 증가할수록 데이터 포인트 간의 거리가 기하급수적으로 멀어지게 되고 희소한 구조를 가지게 됨.
피처가 많아지면 상대적으로 적은 차원에서 학습된 모델보다 예측 신뢰도가 떨어지고 피처가 많을 경우 개별 피처간 상관관계가 높을 가능성이 있어 다중 공선성 문제가 발생할 수 있음.

차원 축소를 통해 피처 수를 줄이면 더 직관적으로 데이터를 해석할 수 있고 3차원 이하의 차원 축소를 통해 시각적으로 데이터를 압축해서 표현할 수도 있음.
차원 축소를 할 경우 학습 데이터의 크기가 줄어들어서 학습에 필요한 처리 능력도 줄일 수 있음.

#01 차원 축소 개요

차원 축소

피처 선택 (Feature Selection) : 종속성이 강한 불필요한 피처는 아예 제거하고 데이터의 특징을 잘 나타내는 주요 피처만 선택하는 것

피처 추출 (Feature Extraction) : 기존 피처를 저차원의 중요 피처로 압축해서 추출하는 것

-> 매우 많은 차원을 가지고 있는 이미지나 텍스트에서 잠재적인 의미를 찾아주기 위해 차원 축소 알고리즘을 많이 사용함.

#01 차원 축소 개요

차원의 저주

데이터 학습을 위해 차원이 증가하면서 학습데이터 수가 차원의 수보다 적어져 성능이 저하되는 현상.

차원이 증가할 수록 개별 차원 내 학습할 데이터 수가 적어지는(sparse) 현상 발생

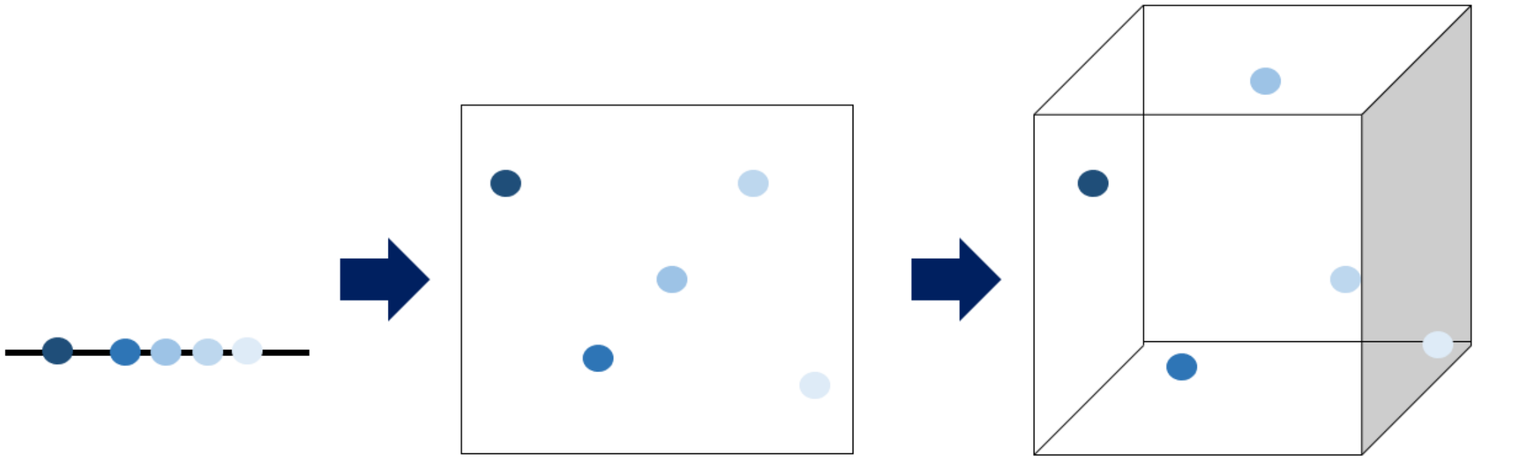
*해결책: 차원을 줄이거나(축소시키거나) 데이터를 많이 획득

-> 차원이 증가함에 따라(=변수의 수 증가) 모델의 성능이 안 좋아지는 현상을 의미

무조건 변수의 수가 증가한다고 해서 차원의 저주 문제가 있는 것이 아니라, 관측치 수보다 변수의 수가 많아지면 발생. ex) 관측치 개수는 200개인데, 변수는 7000개

#01 차원 축소 개요

차원의 저주



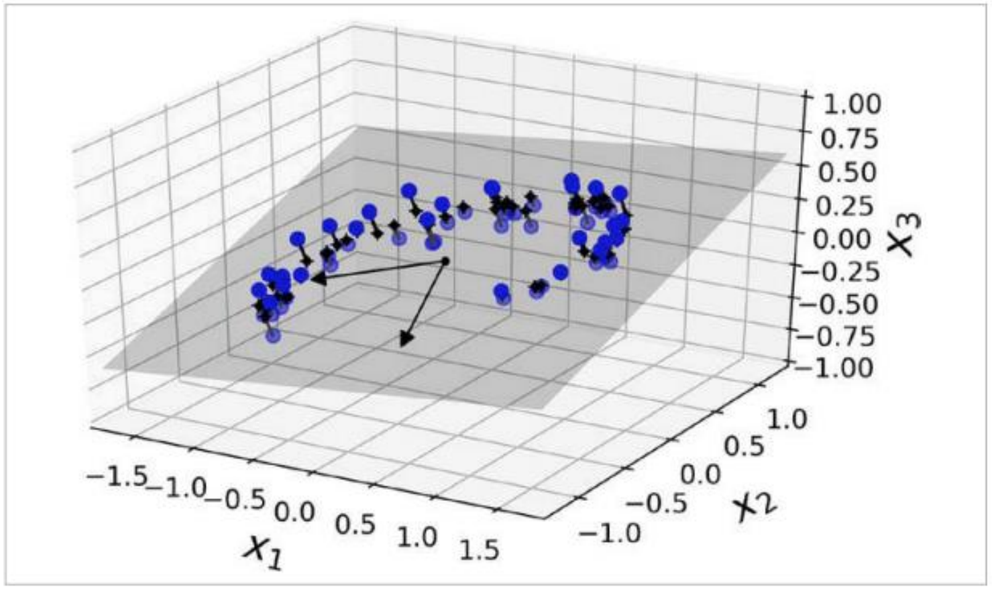
Made by: ta-daa

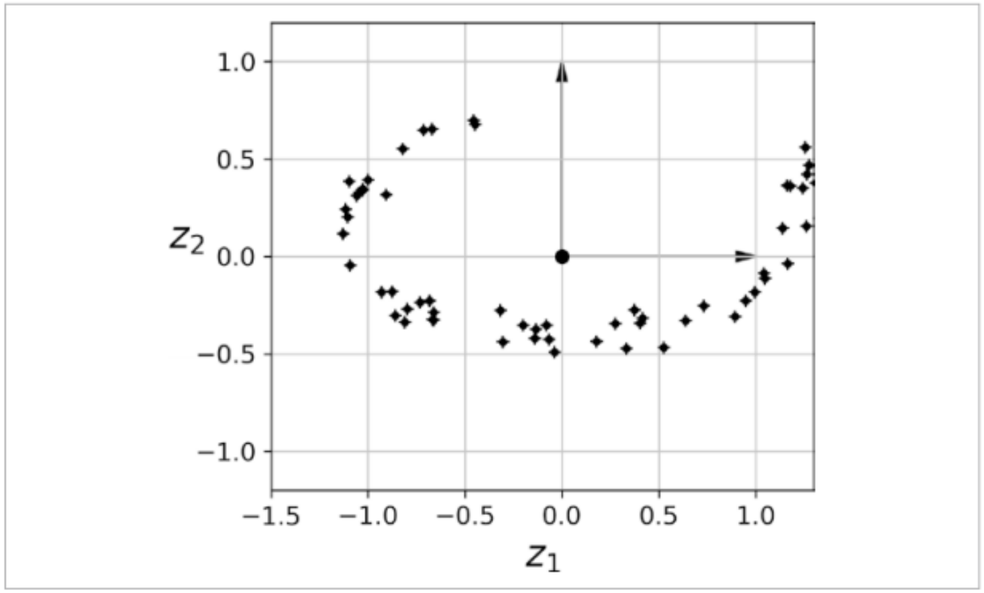
차원이 증가함에 따라, 빈 공간이 생기는 것을 차원의 저주라고 함

#02 차원 축소를 위한 접근 방법

투영

고차원 데이터를 저차원 공간으로 투영하는 방법

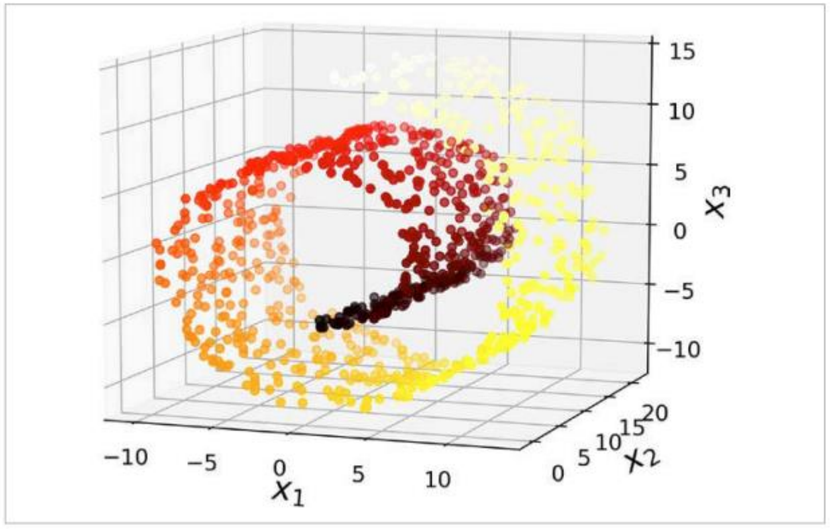


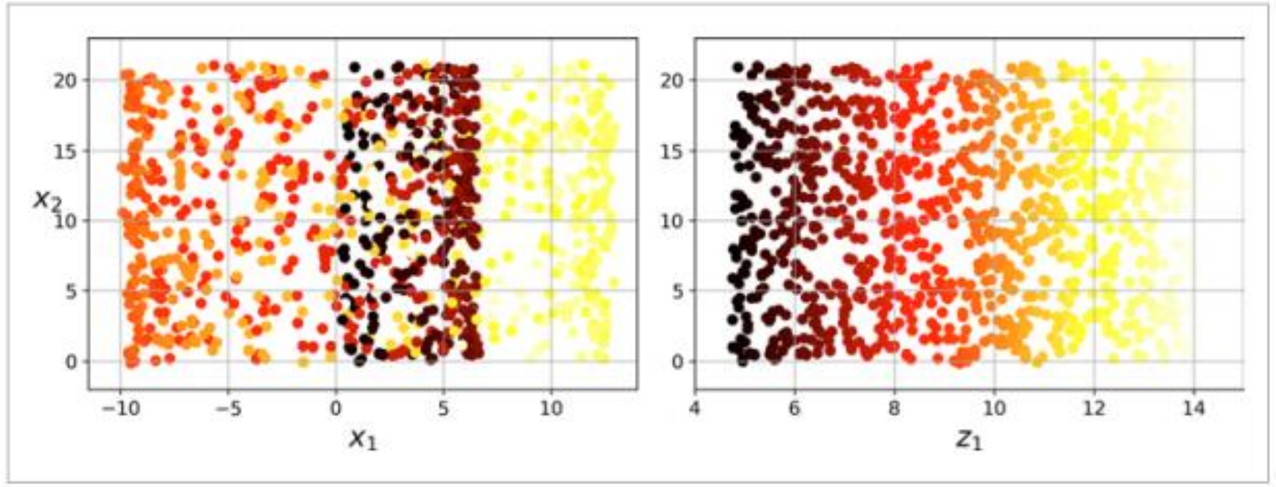


#02 차원 축소를 위한 접근 방법

투영

고차원 데이터를 저차원 공간으로 투영하는 방법

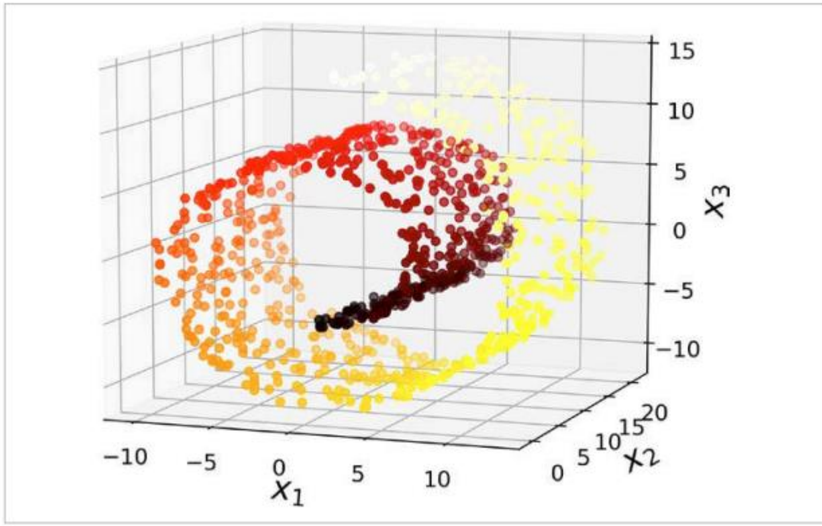




#02 차원 축소를 위한 접근 방법

매니폴드 학습

고차원 데이터가 매니폴드(manifold) 위에 있다고 가정하고, 매니폴드를 저차원 공간으로 매핑하는 방법. 데이터의 구조를 보존하는 것을 목표로 한다.

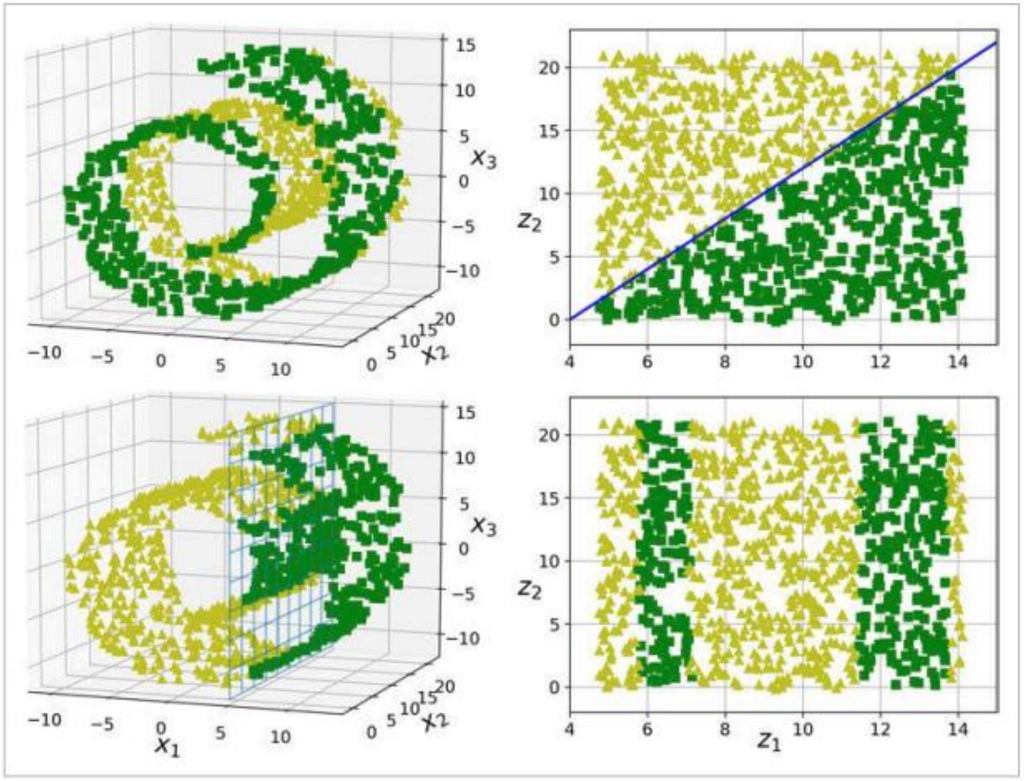


#02 차원 축소를 위한 접근 방법

매니폴드 학습

많은 차원 축소 알고리즘은 이러한 꼬여있는 매니폴드를 풀어헤친 형태를 모델링하는 식으로 작동하는데, 이를 매니폴드 학습이라고 함.

매니폴드 학습이 많이 활용되는 가장 큰 이유는 Classification이나 Regression같은 작업 시 저차원 매니폴드 형태로 데이터를 표현하면 훨씬 더 간단해질 거라고 가정하기 때문.

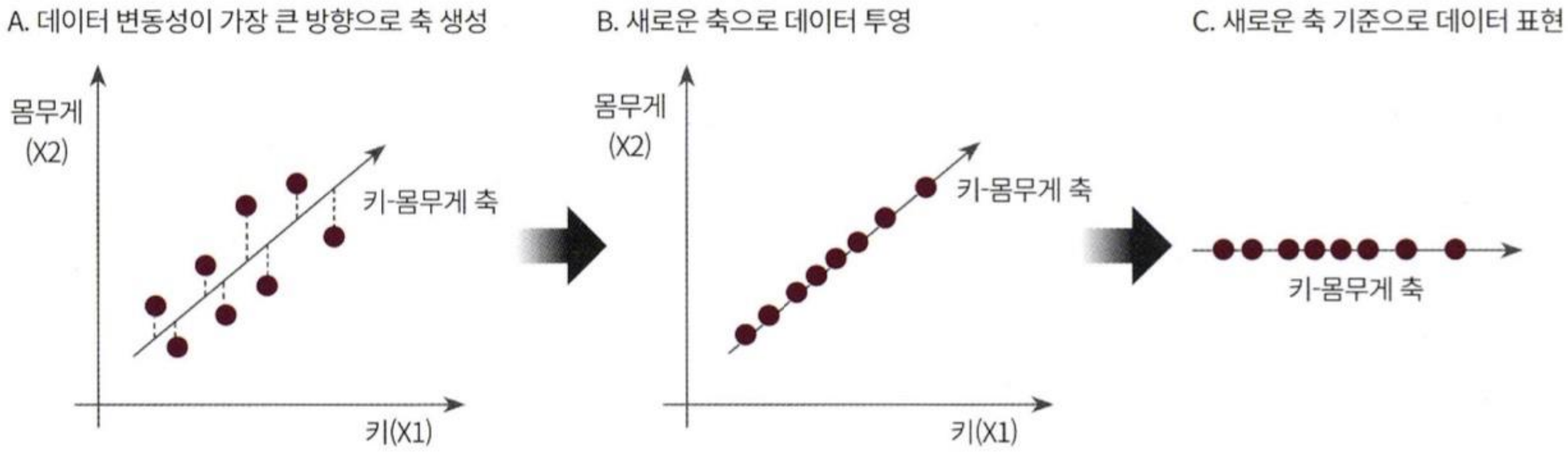


#03 PCA (Principal Component Analysis)

PCA

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

가장 높은 분산을 가지는 데이터의 축을 찾아 이 축으로 차원을 축소하는데 이것이 PCA의 주성분이 됨.



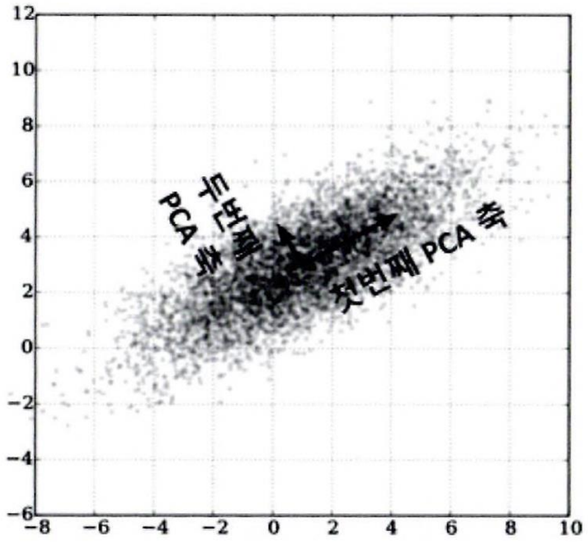
2개의 피처를 한 개의 주성분을 가진 데이터 세트로 차원 축소를 할 수 있음.

데이터 변동성이 가장 큰 방향으로 축을 생성하고, 새롭게 생성된 축으로 데이터를 투영하는 방식.

#03 PCA (Principal Component Analysis)

PCA의 차원 축소

1. 가장 큰 데이터 변동성(Variance)을 기반으로 첫 번째 벡터 축 생성
2. 이 벡터 축에 직각이 되는 벡터(직교 벡터)를 두 번 째 벡터 축으로 설정
3. 다시 두 번째 축과 직각이 되는 벡터를 설정 하는 방식으로 세 번째 축 생성



-> 이렇게 생성된 벡터 축에 원본 데이터를 투영하면 벡터 축의 개수 만큼 원본 데이터가 차원 축소됨

#03 PCA (Principal Component Analysis)

선형대수 관점에서의 PCA

입력 데이터의 공분산 행렬을 고유값 분해 후 구한 고유벡터에 입력데이터를 선형 변환하는 것

1. 입력 데이터 세트의 공분산 행렬 생성
2. 공분산 행렬의 고유벡터와 고유값 계산
3. 고유값이 가장 큰 순으로 K개(PCA 변환 차수만큼)만큼 고유벡터 추출
4. 고유값이 가장 큰 순으로 추출된 고유벡터를 이용해 새롭게 입력 데이터 변환

-> PCA는 많은 속성으로 구성된 원본 데이터를 그 핵심을 구성하는 데이터로 압축한 것

#03 PCA (Principal Component Analysis)

sepal length, sepal width, petal length, petal width 이 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 pit
%matplotlib inline

iris = load_iris()
# 넘파이 데이터 세트를 판다스 DataFrame으로 변환
columns = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']
irisDF = pd.DataFramedris.data, columns=columns)
irisDF['target']=iris.target
irisDF.head(3)
```

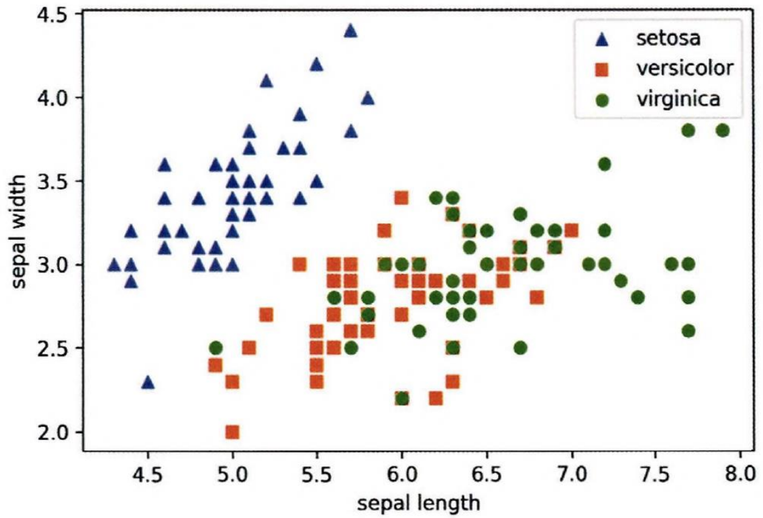
	sepal_length	sepal_width	petal_length	petal_width	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0

#03 PCA (Principal Component Analysis)

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

sepal length, sepal width를 각 X축, Y축으로 품종 데이터 분포를 나타냄

```
#setosa는 세모, versicolor는 네모, virginica는 동그라미로 표현
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']
    pit.scatter(x_axis_data, y_axis_data, marker=marker,
label=iris.target_names[i])

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



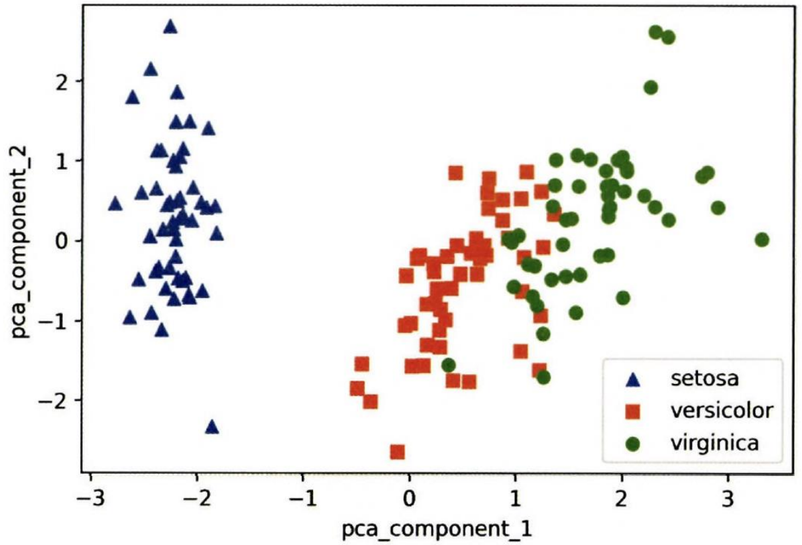
Setosa 품종의 경우 sepal width가 3.0보다 크고, sepal length가 6.0 이하인 곳에 일정하게 분포돼 있음
Versicolor와 virginica의 경우는 sepal width와 sepal length 조건만으로는 분류가 어려운 복잡한 조건임을 알 수 있다

#03 PCA (Principal Component Analysis)

PCA를 통해 4개 속성을 2개로 압축한 뒤 2개의 PCA 속성으로 붓꽃 데이터의 품종 분포 시각화

-> PCA는 여러 속성의 값을 연산해야 하므로 속성의 스케일에 영향을 받기에 여러 속성을 PCA로 압축 하기 전에 각 속성값을 동일한 스케일로 변환하는 과정 필요

	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



pca_component_10이 원본 데이터의 변동성을 잘 반영 했기에 PCA 변환 후에도 pca_component_1 축을 기반으로 setosa 품종을 명확하게 구분 가능

Versicolor, Viriginica는 pca_component_1 축을 기반으로 서로 겹치는 부분이 일부 존재하지만 비교적 잘 구분됨

#03 PCA (Principal Component Analysis)

print(pca.explained_variance_ratio_)

> [0.72962445 0.22850762]

첫 번째 PCA 변환 요소인 pca_component_1이 전체 변동성의 약 72.9%를 차지
두 번째인 pca_component_2가 약 22.8%를 차지
그렇기에 PCA를 2개 요소로만 변환해도 원본 데이터 의 변동성을 95% 설명할 수 있음

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

> 원본 데이터 교차 검증 개별 정확도:[0.98 0.94 0.96]
원본 데이터 평균 정확도:0.96

> PCA 변환 데이터 교차 검증 개별 정확도:[0.88 0.88 0.88]
PCA 변환 데이터 평균 정확도:0.88

원본 데이터 세트 대비 예측 정확도는 PCA 변환 차원 개수에 따라 예측 성능이 떨어질 수 밖에 없음

8%의 정확도 하락은 비교적 큰 성능 수치의 감소지만 속성 개수가 50% 감소한 것을 고려한다면

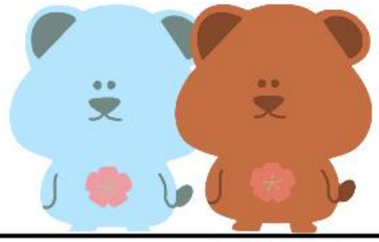
PCA 변환 후에도 원본 데이터의 특성을 상당 부분 유지 하고 있음

LDA & SVD

2조 김정은

EWHA EURON

목차



EWHA EURON

#01 적절한 차원 수 선택

#1 적절한 차원 수 선택

➔ 축소할 차원 수를 임의로 정하기보다는 충분한 분산(ex 95%)이 될 때까지 더해야 할 차원 수를 선택하는 것이 간단, 데이터 시각화를 위해 차원을 축소하는 경우, 2개나 3개로 줄이는 것이 일반적

➔ 예를 들면, 차원을 축소하지 않고 PCA를 계산한 뒤 훈련 세트의 분산을 95%로 유지하는 데 필요한 최소한의 차원 수를 계산하는 방법

➔ 또는 설명된 분산을 차원 수에 대한 함수로 그리는 방법(cumsum을 그래프화)
➔ 설명된 분산의 빠른 성장이 멈추는 변곡점이 존재.
➔ 여기서는 차원을 약 100으로 축소해도 설명된 분산이 손상되지 않을 것

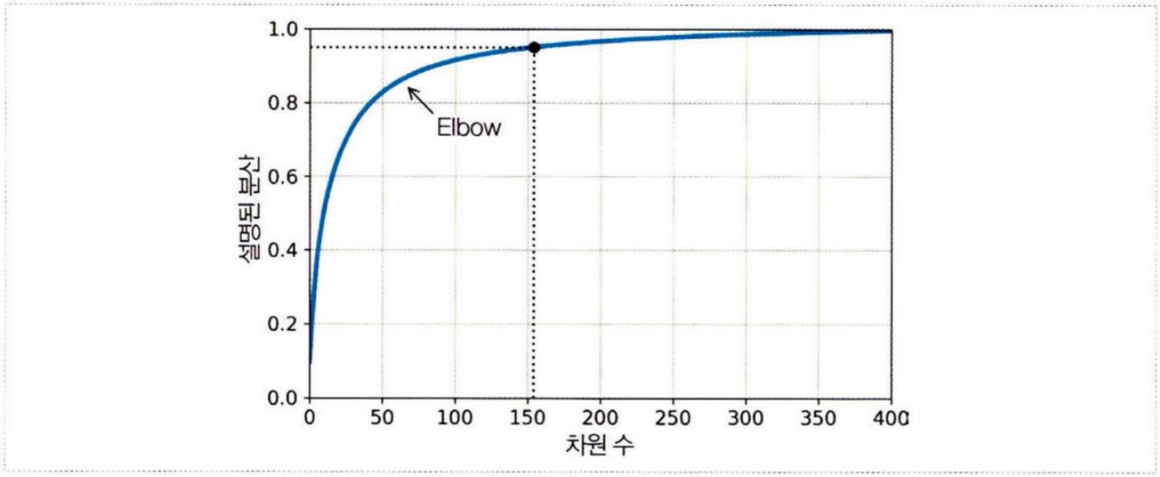


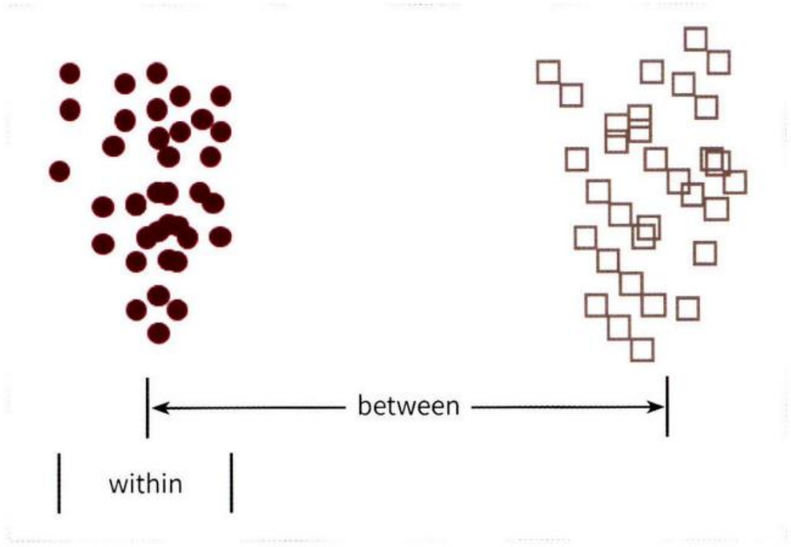
그림 8-8 차원 수에 대한 함수로 나타낸 설명된 분산

#02 LDA

#1 LDA(Linear Discriminant Analysis)

➔ 선형 판별 분석법, PCA와 유사
➔ PCA와 유사하게 입력 데이터 세트를 저차원 공간에 투영하여 차원을 축소
➔ 지도학습의 분류(Classification)에서 사용하기 쉽도록 개별 클래스를 분별할 수 있는 기준을 최대한 유지, 차원을 축소
➔ PCA : 입력 데이터의 변동성의 가장 큰 축 탐색
➔ LDA : 입력 데이터의 결정 값 클래스를 최대한으로 분리할 수 있는 축 탐색
➔ 클래스 간 분산과 클래스 내부 분산의 비율을 최대화하는 방식으로 차원을 축소
➔ 클래스 간 분산은 최대 / 클래스 내부 분산은 최소

PCA와 **다른 점은,**
공분산 행렬이 아니라
클래스 간 분산과 클래스 내부 분산 행렬을 생성 후
이 행렬에 기반한 고유 벡터를 구하고
입력 데이터를 투영



#02 LDA

#2 LDA를 구하는 STEP

1. 클래스 내부와 클래스 간 분산 행렬을 구한다.
이 두 개의 행렬은 입력 데이터의 결정 값 클래스별로, 개별 피처의 평균 벡터(mean vector)을 기반으로 구한다.

2. 클래스 내부 분산 행렬을 S_W, 클래스 간 분산 행렬을 S_B라고 하면 다음 식으로 두 행렬을 고유 벡터로 분해 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 순서대로 K개(LDA 변환 차수만큼) 추출

4. 고유값이 가장 큰 순서대로 추출된 고유 벡터를 이용해 새롭게 입력 데이터를 변환

#02 LDA

#3 붓꽃 데이터 세트에 LDA 적용

➔ 사이킷런의 LDA를 이용해 변환, 그 결과를 품종별로 시각화

➔ 사이킷런은 LDA를 LinearDiscriminantAnalysis 클래스로 제공

In [1]:
```
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)
```

➔ 붓꽃 데이터 세트를 로드 후 표준 정규 분포로 스케일링

➔ 2개의 컴포넌트로 붓꽃 데이터를 LDA 변환할 예정

➔ PCA와는 달리, LDA는 지도학습!

➔ 클래스의 결정 값이 변환 시 필요

In [2]:
```
lda = LinearDiscriminantAnalysis(n_components=2)
lda.fit(iris_scaled, iris.target)
iris_lda = lda.transform(iris_scaled)
print(iris_lda.shape)
```

```
(150, 2)
```

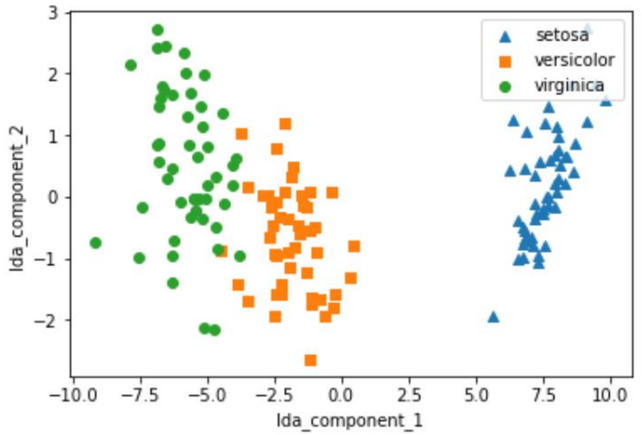
➔ lda 객체의 fit( ) 메서드를 호출할 때 결정값이 입력됨

EWHA EURON

#02 LDA

#3 붓꽃 데이터 세트에 LDA 적용

In [5]:
```
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 모양을 산점도로 표시
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()
```



➔ LDA 변환된 입력 데이터 값을 2차원 평면에 품종별로 표현, 소스 코드는 PCA와 큰 차이 X

➔ PCA로 변환된 데이터와 좌우 대칭 형태로 많이 닮아 있음

EWHA EURON

#03 SVD

#1 SVD(Singular Value Decomposition)

➔ PCA와 유사한 행렬 분해 기법

➔ PCA : 정방 행렬만을 고유 벡터로 분해 가능 / SVD : 정방 행렬뿐 아니라 다른 행렬에도 적용 가능

➔ 일반적으로 SVD는 m x n 크기의 행렬을 다음과 같이 분해

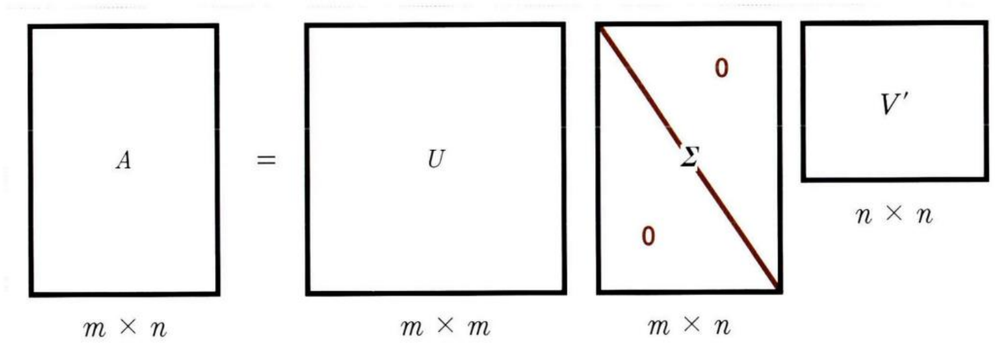
$$A = U \sum V^T$$

➔ SVD는 특이값 분해로 불리며, 행렬 U와 V에 속한 벡터는 특이 벡터 -> 모든 특이 벡터는 직교

➔ Σ는 대각 행렬, 행렬의 대각에 위치한 값만 0이 아니고 나머지 위치의 값들은 모두 0

➔ Σ가 위치한 0이 아닌 값이 바로 행렬 A의 특이값

➔ SVD는 A의 차원이 m x n 일 때, U의 차원이 m x m, Σ의 차원이 m x n, V^T의 차원이 n x n 으로 분해



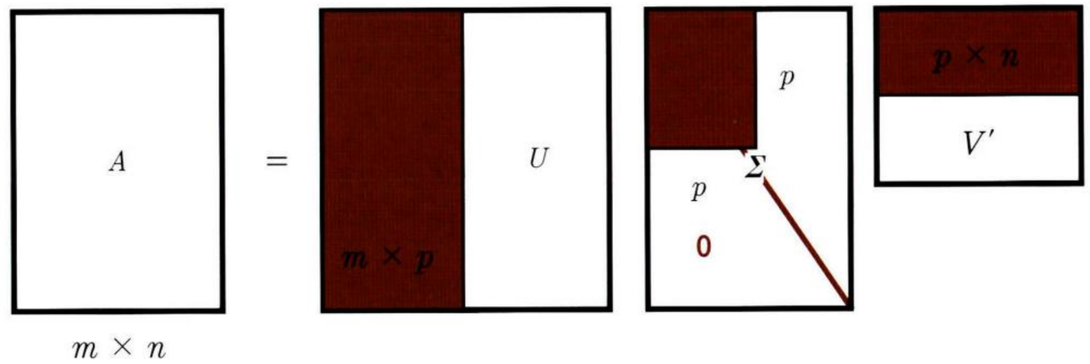
#03 SVD

#1 SVD(Singular Value Decomposition)

➔ 일반적으로는 Σ의 비대각인 부분과 대각 원소 중 특이값이 0인 부분도 모두 제거

➔ 제거된 Σ에 대응되는 U와 V 원소도 함께 제거해 차원을 줄인 형태로 SVD 적용

➔ SVD 적용 시 -> A의 차원이 m x n 일 때, U의 차원을 m x p, Σ의 차원을 p x p, V^T의 차원을 p x n 으로 분해



➔ Truncated SVD는 Σ의 대각 원소 중 상위 몇 개만 추출, 대응하는 U와 V의 원소도 함께 제거

➔ 더욱 차원을 줄인 형태로 분해

➔ 일반적으로 SVD는 넘파이나 사이파이 라이브러리를 이용하여 수행

#03 SVD

#2 넘파이 SVD 연산

➔ 랜덤한 4 x 4 넘파이 행렬을 생성
➔ **행렬의 개별 로우끼리의 의존성을 없애기 위해 랜덤 행렬 생성**
➔ 생성된 a 행렬에 SVD 적용 -> **U, Sigma, V_t**를 도출
➔ SVD 분해는 numpy.linalg.svd 에 파라미터로 원본 행렬을 입력하면 U, Sigma, V 행렬을 반환
➔ Sigma 행렬의 경우, A = UΣV^T에서 Σ행렬을 나타냄
➔ Σ행렬의 경우 행렬의 대각에 위치한 값만 0이 아니고 그렇지 않은 경우는 모두 0
➔ 0이 아닌 값의 경우만 1차원 행렬로 표현

```
In [7]: 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))
```

```
(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 행렬이 4 x 4 행렬로 반환
➔ V_t 행렬이 4 x 4 행렬로 반환
➔ Sigma는 (4,) 1차원 행렬로 반환
➔ 내적하면 다시 원본 행렬로 복원

EWHA EURON

#03 SVD

#2 넘파이 SVD 연산

➔ 데이터 세트가 로우 간 의존성이 있을 경우

➔ 의존성의 부여를 위해 a 행렬의 3번째 로우를 '1번째 로우 + 2번째 로우 ' 로 업데이트, 4번째 로우는 1번째 로우와 똑같이 업데이트

```
In [9]: a[2] = a[0] + a[1]
        a[3] = a[0]
        print(np.round(a, 3))
```

```
[[-0.212 -0.285 -0.574 -0.44 ]
 [-0.33   1.184  1.615  0.367]
 [-0.542  0.899  1.041 -0.073]
 [-0.212 -0.285 -0.574 -0.44 ]]
```

➔ a 행렬 -> 로우 간 관계가 매우 높아짐

```
In [10]: # 다시 SVD를 수행해 Sigma 값 확인
         U, Sigma, Vt = svd(a)
         print(U.shape, Sigma.shape, Vt.shape)
         print('Sigma Value:₩n', np.round(Sigma, 3))
```

```
(4, 4) (4,) (4, 4)
Sigma Value:
 [2.663 0.807 0.    0.   ]
```

➔ 이전과 차원은 같지만 Sigma 값 중 2개가 0으로 변함

➔ 선형 독립인 로우 벡터의 수가 2개(행렬의 rank 2)

➔ 분해된 U, Sigma, V_t를 이용해 다시 원본 행렬로 복원 가능

➔ U, Sigma, V_t의 전체 데이터 대신 Sigma의 0에 대응되는 U, Sigma, V_t의 데이터를 제외하고 복원

#03 SVD

#3 Truncated SVD

➔ Σ행렬에 있는 대각 원소, 즉 특이값 중 상위 일부 데이터만 추출해 분해
➔ 이 방식으로 분해하면 더 작은 차원의 U, Σ, V^T로 분해하기 때문에 원본 행렬을 정확히 복원 불가
➔ 상당한 수준으로 '근사' 까지만 가능
➔ 원래 차원의 차수에 가깝게 잘라낼수록(Truncate) 원본 행렬에 더 가깝게 복원 가능

In [11]:
```
import numpy as np
from scipy.sparse.linalg import svds
from scipy.linalg import svd

# 원본 행렬을 출력하고 SVD를 적용할 경우 U, Sigma, Vt의 차원 확인
np.random.seed(121)
matrix = np.random.random((6, 6))
print('원본 행렬:₩n', matrix)
U, Sigma, Vt = svd(matrix, full_matrices=False)
print('₩n분해 행렬 차원:', U.shape, Sigma.shape, Vt.shape)
print('₩nSigma값 행렬:', Sigma)

# Truncated SVD로 Sigma 행렬의 특이값을 4개로 하여 Truncated SVD 수행
num_components = 4
U_tr, Sigma_tr, Vt_tr = svds(matrix, k=num_components)
print('₩nTruncated SVD 분해 행렬 차원:', U_tr.shape, Sigma_tr.shape, Vt_tr.shape)
print('₩nTruncated SVD Sigma값 행렬:', Sigma_tr)
matrix_tr = np.dot(np.dot(U_tr, np.diag(Sigma_tr)), Vt_tr) # output of TruncatedSVD

print('₩nTruncated SVD로 분해 후 복원 행렬:₩n', matrix_tr)
```

➔ Truncated SVD는 사이파이에서만 지원(희소 행렬로만 지원, scipy.sparse.linalg.svds 이용)

#03 SVD

#3 Truncated SVD

➔ 6 x 6 행렬을 SVD 분해하면 U, Sigma, V_t가 각각 ((6, 6), (6,), (6, 6)) 차원

➔ 그러나 Truncated SVD의 n_components를 4로 설정해 U, Sigma, V_t를 ((6, 4), (4,), (4, 6))로 분해

➔ Truncated SVD로 분해된 행렬로 다시 복원할 경우 완벽하게 복원되지 않고 근사적으로 복원

```
원본 행렬:
 [[0.11133083 0.21076757 0.23296249 0.15194456 0.83017814 0.40791941]
 [0.5557906  0.74552394 0.24849976 0.9686594  0.95268418 0.48984885]
 [0.01829731 0.85760612 0.40493829 0.62247394 0.29537149 0.92958852]
 [0.4056155  0.56730065 0.24575605 0.22573721 0.03827786 0.58098021]
 [0.82925331 0.77326256 0.94693849 0.73632338 0.67328275 0.74517176]
 [0.51161442 0.46920965 0.6439515  0.82081228 0.14548493 0.01806415]]

분해 행렬 차원: (6, 6) (6,) (6, 6)

Sigma값 행렬: [3.2535007  0.88116505 0.83865238 0.55463089 0.35834824 0.0349925 ]

Truncated SVD 분해 행렬 차원: (6, 4) (4,) (4, 6)

Truncated SVD Sigma값 행렬: [0.55463089 0.83865238 0.88116505 3.2535007 ]

Truncated SVD로 분해 후 복원 행렬:
 [[0.19222941 0.21792946 0.15951023 0.14084013 0.81641405 0.42533093]
 [0.44874275 0.72204422 0.34594106 0.99148577 0.96866325 0.4754868 ]
 [0.12656662 0.88860729 0.30625735 0.59517439 0.28036734 0.93961948]
 [0.23989012 0.51026588 0.39697353 0.27308905 0.05971563 0.57156395]
 [0.83806144 0.78847467 0.93868685 0.72673231 0.6740867  0.73812389]
 [0.59726589 0.47953891 0.56613544 0.80746028 0.13135039 0.03479656]]
```

#03 SVD

#4 사이킷런 TruncatedSVD 클래스

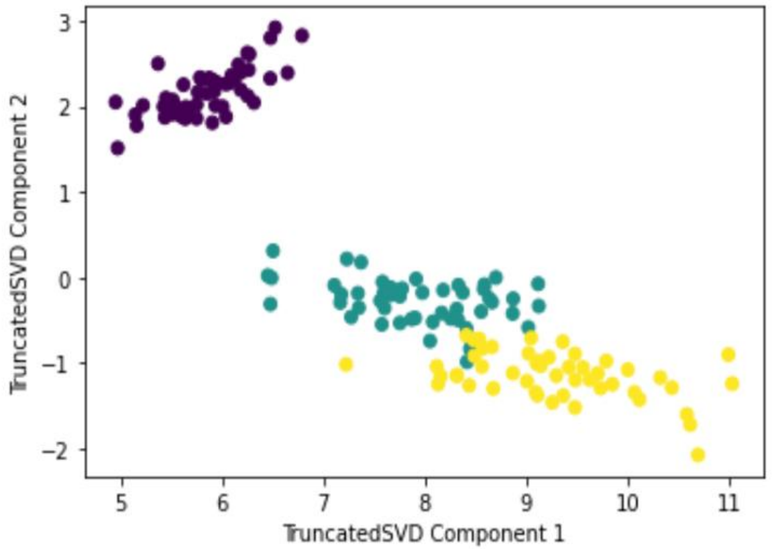
➔ 사이킷런 TruncatedSVD 클래스는 사이파이와 svds와 같이 Truncated SVD 연산 수행 후 원본 행렬을 분해한 U, Sigma, V_t 행렬을 반환하지 않음

➔ PCA 클래스와 유사하게 fit( )와 transform( ) 을 호출해 원본 데이터를 몇 개의 주요 컴포넌트(K개)로 차원을 축소해 변환

➔ 원본 데이터를 Truncated SVD 방식으로 분해된 U*Sigma 행렬에 선형 변환해 생성

In [12]:
```
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 컴포넌트로 TruncatedSVD 변환
tsvd = TruncatedSVD(n_components=2)
tsvd.fit(iris_ftrs)
iris_tsvd = tsvd.transform(iris_ftrs)

# 산점도 2차원으로 TruncatedSVD 변환된 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')
```

Out[12]: Text(0, 0.5, 'TruncatedSVD Component 2')

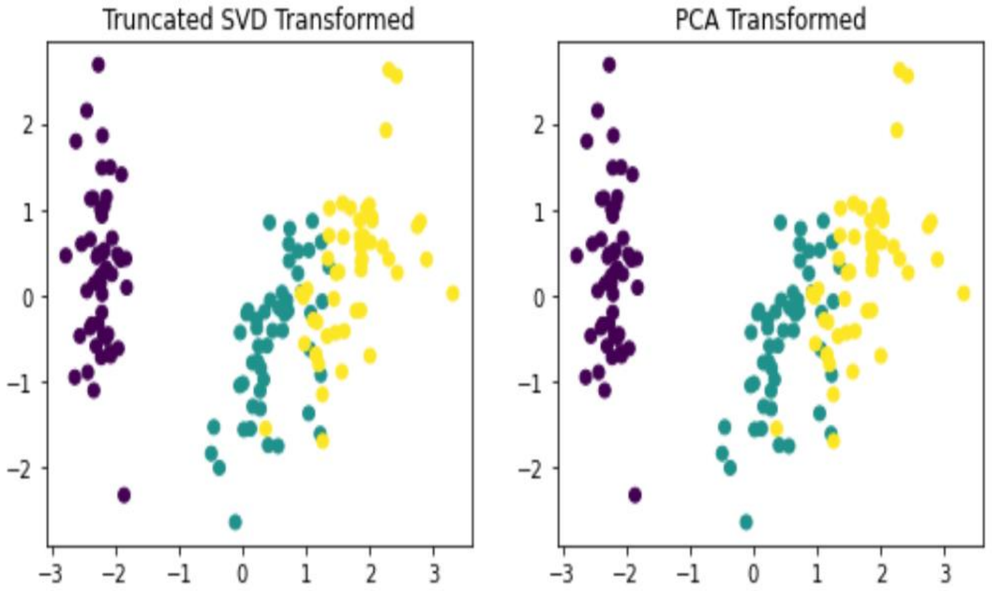


#03 SVD

#4 사이킷런 TruncatedSVD 클래스

➔ TruncatedSVD 변환 역시 PCA와 유사하게 변환 후에 품종별로 어느 정도 클러스터링이 가능함
➔ 각 변환 속성으로 뛰어난 고유성을 가짐
➔ TruncatedSVD와 PCA 클래스 모두 SVD를 이용해 행렬을 분해
➔ 붓꽃 데이터를 스케일링으로 변환 후 TruncatedSVD와 PCA 클래스 변환을 하면 두 개가 거의 동일

```
Out[13]: Text(0.5, 1.0, 'PCA Transformed')
```

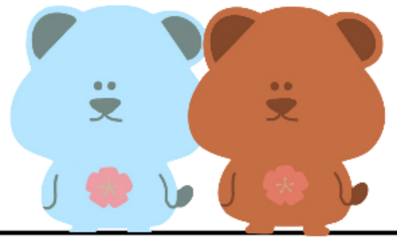



```
print((iris_pca - iris_tsvd).mean())
print((pca.components_ - tsvd.components_).mean())
```

```
2.363664819426958e-15
-6.461844948013606e-17
```

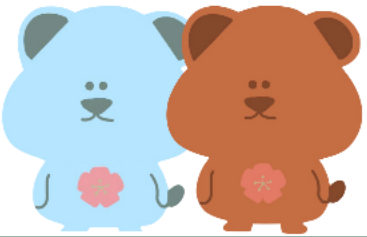
➔ 모두 0에 가까운 값
➔ 2개의 변환 서로 동일

THANK YOU



EWHA EURON

6.5 NMF와 Topic Modeling



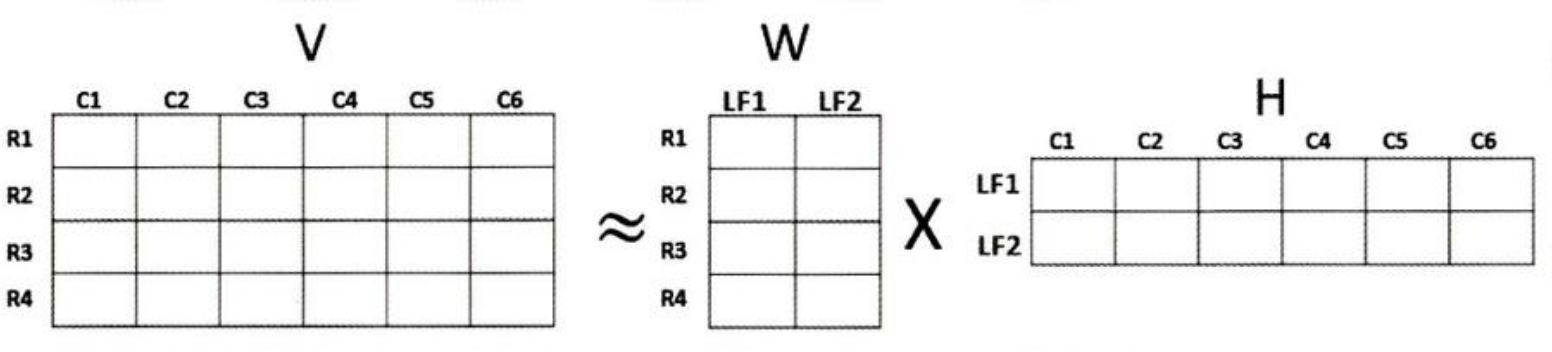
초급 세션 2팀 조주현

EWHA EURON

#1. NMF (Non-Negative Matrix Factorization)

>> NMF이란?

원본 행렬 내의 모든 원소 값이 모두 양수일 때 다음과 같이 좀 더 간단하게 두개의 기반 양수 행렬로 분해될 수 있는 기법



>> 행: 샘플 열: feature

>> 특징 추출, 차원 축소 → 행렬 W

>> 주제 찾기 → 행렬 H

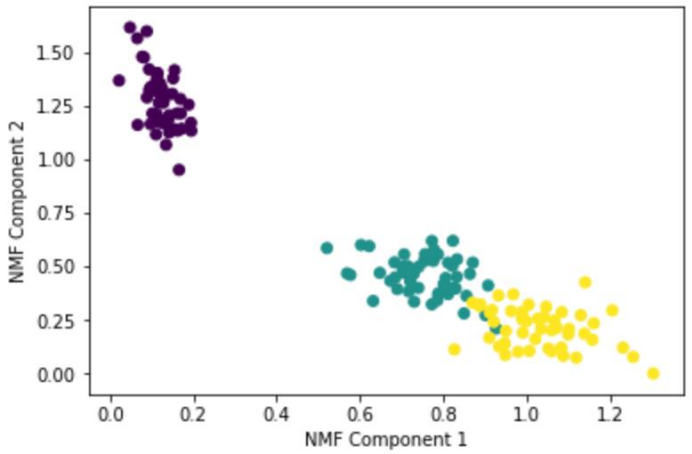
>> W: 인덱스의 행에 특성이 얼마나 적합한가

>> H: 열이 특성에 얼마나 중요한가

#1. 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')
```

```
: Text(0,0.5,'NMF Component 2')
```



#2. 토픽 모델링 (Topic Modeling)

>> 토픽 모델링이란?

문서 집합에서 중요 주제를 효과적으로 적은 시간 안에 찾아내는 것

>> 중심 단어 함축적 추출

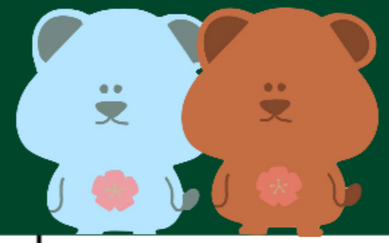
>> LDA (Latent Dirichlet Allocation) 이용

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation

# 모토사이클, 야구, 그래픽스, 윈도우즈, 중동, 기독교, 전자공학, 의학 등 8개 주제를 추출.
cats = ['rec.motorcycles', 'rec.sport.baseball', 'comp.graphics', 'comp.windows.x',
        'talk.politics.mideast', 'soc.religion.christian', 'sci.electronics', 'sci.med'  ]

# 위에서 cats 변수로 기재된 category만 추출. featch_20newsgroups( )의 categories에 cats 입력
news_df = fetch_20newsgroups(subset='all',remove=('headers', 'footers', 'quotes'),
                             categories=cats, random_state=0)

#LDA 는 Count기반의 Vectorizer만 적용합니다.
count_vect = CountVectorizer(max_df=0.95, max_features=1000, min_df=2, stop_words='english', ngram_range=(1,2))
feat_vect = count_vect.fit_transform(news_df.data)
print('CountVectorizer Shape:', feat_vect.shape)
```

```
CountVectorizer Shape: (7862, 1000)
```

#2. 토픽 모델링 (Topic Modeling)

```
def display_topic_words(model, feature_names, no_top_words):
    for topic_index, topic in enumerate(model.components_):
        print('₩nTopic #',topic_index)

        # components_ array에서 가장 값이 큰 순으로 정렬했을 때, 그 값의 array index를 반환.
        topic_word_indexes = topic.argsort()[::-1]
        top_indexes=topic_word_indexes[:no_top_words]

        # top_indexes대상인 index별로 feature_names에 해당하는 word feature 추출 후 join으로 concat
        feature_concat = ' '.join([str(feature_names[i]) for i in top_indexes])
        #feature_concat = ' + '.join([str(feature_names[i])+'*'+str(round(topic[i],1)) for i in top_indexes])
        print(feature_concat)

# CountVectorizer객체내의 전체 word들의 명칭을 get_features_names( )를 통해 추출
feature_names = count_vect.get_feature_names()

# Topic별 가장 연관도가 높은 word를 15개만 추출
display_topic_words(lda, feature_names, 15)

# 모토사이클, 야구, 그래픽스, 윈도우즈, 중동, 기독교, 전자공학, 의학 등 8개 주제를 추출.
```

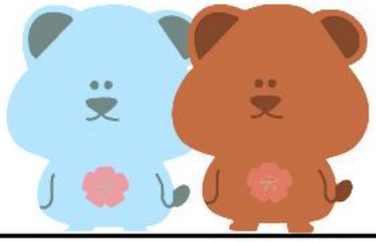


Dimensionality Reduction for Beginners

2팀 이채원

EWHA EURON

목차

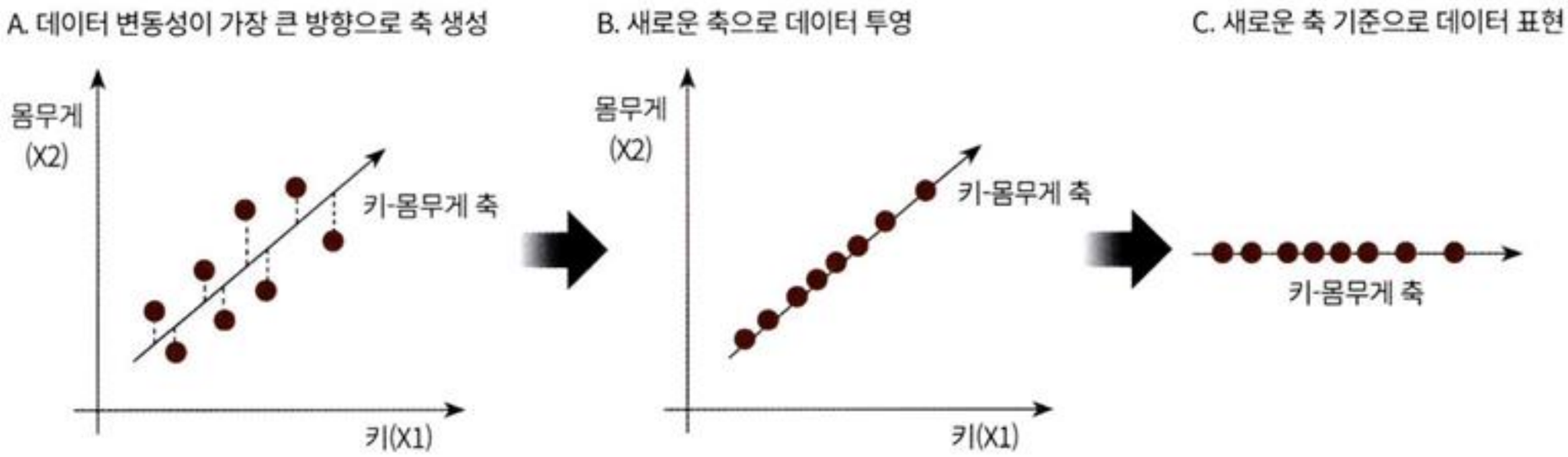


EWHA EURON

#01 PCA

PCA란?

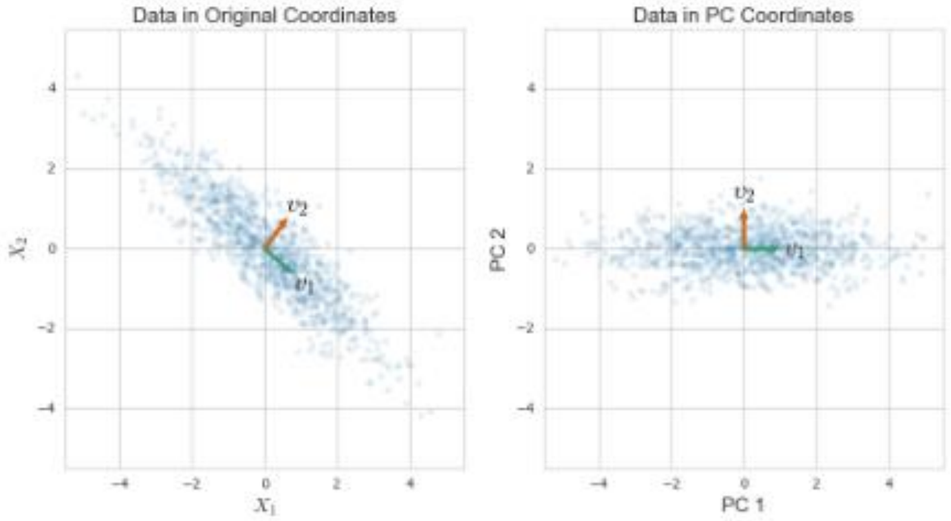
가장 높은 분산을 가진 데이터의 축을 찾아
해당 축으로 차원을 축소하는 차원 축소 기법



#01 PCA

PCA란?

다음과 같은 데이터가 있을 때 :



분산을 가장 잘 보존하는 v1을 **첫 번째 주성분**으로,
첫 번째 벡터와 직교하고 남은 분산을 가장 잘 보존하는 v2를
두 번째 주성분으로 설정.
-> 주성분으로 정의한 평면에 데이터를 투영해 차원 축소!

#01 PCA

사이킷런 : PCA를 이용해 구현 가능

여러 속성의 값을 연산하므로
스케일의 영향을 받음
➔**표준화**

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

canc_norm = StandardScaler().fit(X_canc).transform(X_canc)

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

canc_pca = pca.transform(canc_norm)

print('Number of Features in Breast Cancer Dataset Before PCA : {}\n\nNumber of
  Featrues in Breast Cancer Dataset After PCA: {}'.format(X_canc.shape[1],
 canc_pca.shape[1]))
```

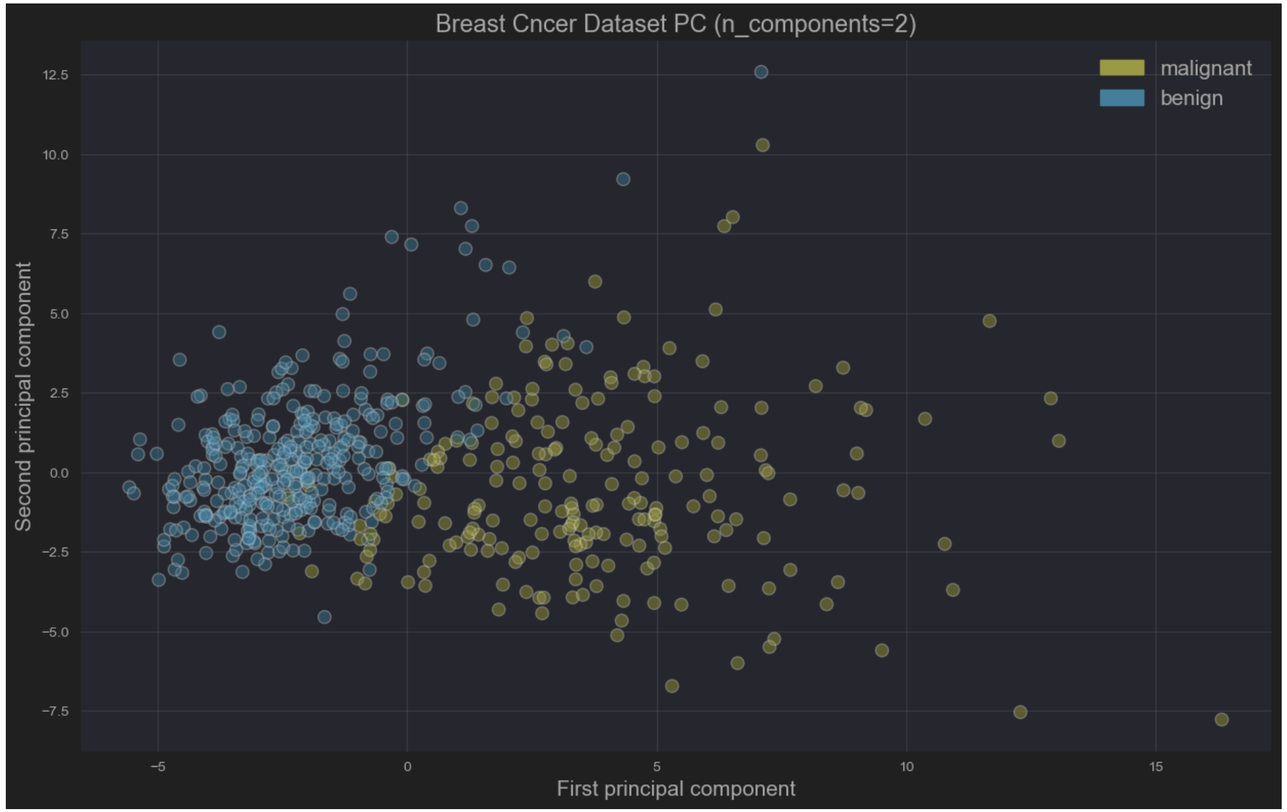
Executed at 2024.05.12 23:52:22 in 889ms

```
Number of Features in Breast Cancer Dataset Before PCA : 30

Number of Featrues in Breast Cancer Dataset After PCA: 2
```

차원 축소 후 피처 수 감소 확인 가능!

#01 PCA

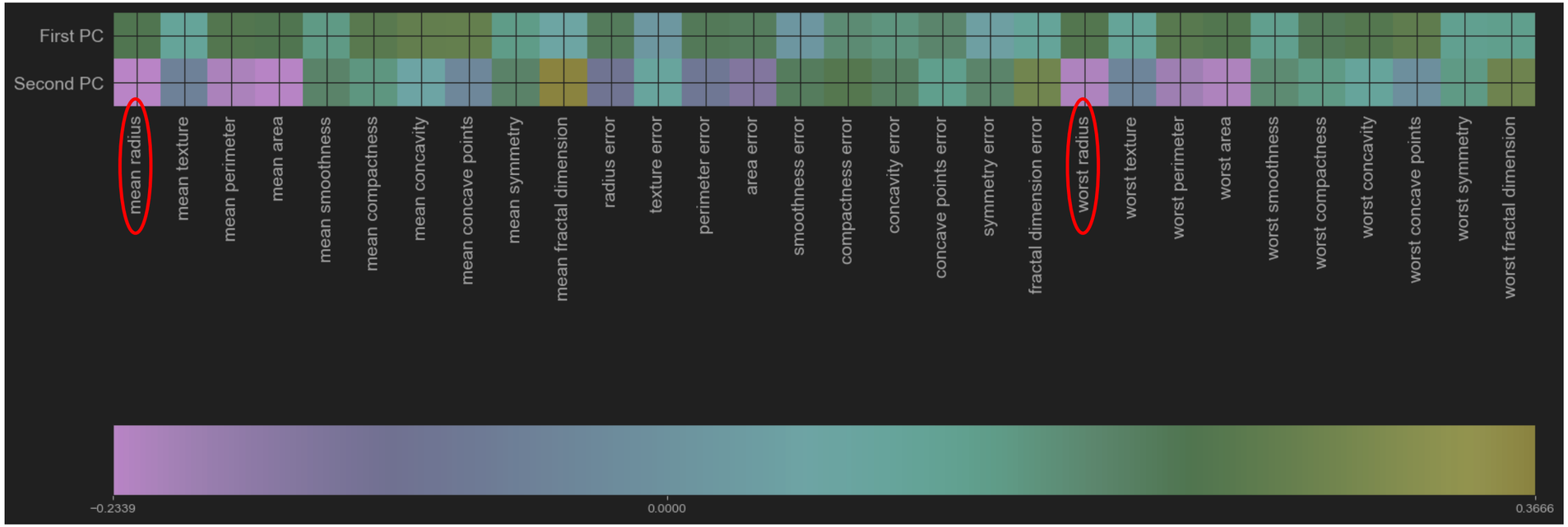
시각화 결과 :



피처의 개수를 2개로 줄이면서
피처 간의 상관관계 확인과 시각화가 더 용이해짐.

#01 PCA

기존 피처와 축소한 피처 간의 상관관계 시각화 결과 :

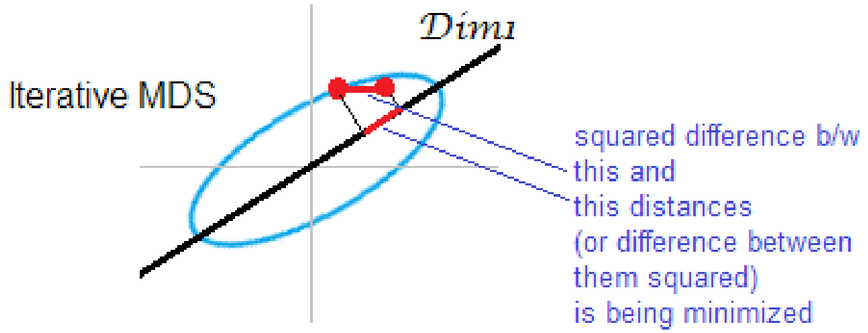


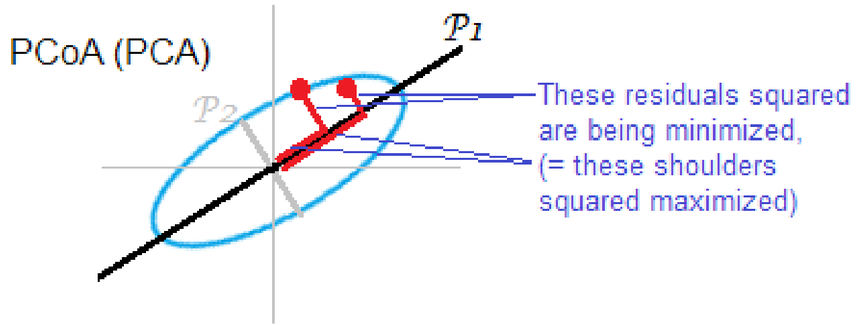
첫 번째 피처 : 모두 양의 상관관계 -> 함께 증가하고 감소
두 번째 피처 : 음의 상관관계가 일부 존재.
유사한 군집이 함께 변화하는 경우 확인 가능 (mean radius, worst radius 등)

#02 MDS

MDS란?

- Manifold Learning Algorithm의 일종 : 고차원 구조를 저차원 구조로 변환
- 기존 데이터 공간에서 두 점의 거리를 보존하여 차원 변환





#02 MDS

사이킷런 : MDS를 이용해 구현 가능

```
from sklearn.manifold import MDS

mds = MDS(n_components= 2, random_state=2)

canc_mds = mds.fit_transform(canc_norm)

print('Number of Features in Breast Cancer Dataset Before MDS : {}\n\nNumber of
  Featrues in Breast Cancer Dataset After MDS: {}'.format(X_canc.shape[1],
 canc_mds.shape[1]))

plot_labelled_scatter(canc_mds, y_canc, ['malignant', 'benign'], (15, 9))

plt.xlabel('First MDS dimension', fontsize=15)
plt.ylabel('Second MDS dimension', fontsize=15)
plt.title('Breast Cacner Dataset MDS (n_components = 2)', fontsize=17)
```

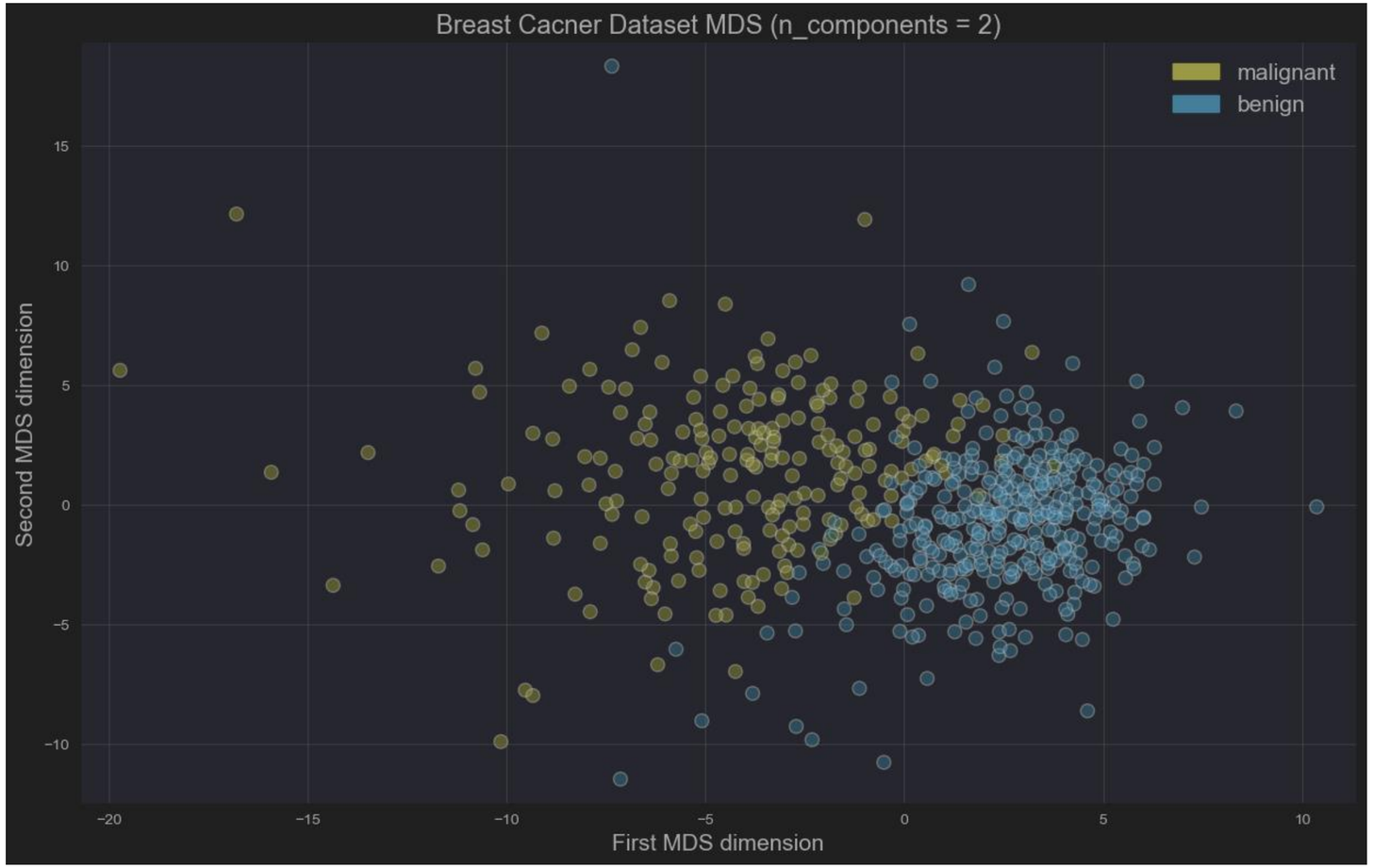
Executed at 2024.05.13 00:12:28 in 17s 822ms

```
C:\Users\chwon\anaconda3\Lib\site-packages\sklearn\manifold\_mds.py:299:
FutureWarning: The default value of `normalized_stress` will change to
`'auto'` in version 1.4. To suppress this warning, manually set the value
of `normalized_stress`.
  warnings.warn(

Number of Features in Breast Cancer Dataset Before MDS : 30

Number of Featrues in Breast Cancer Dataset After MDS: 2
```

차원 축소 후 피처 수 감소 확인 가능!

#02 MDS

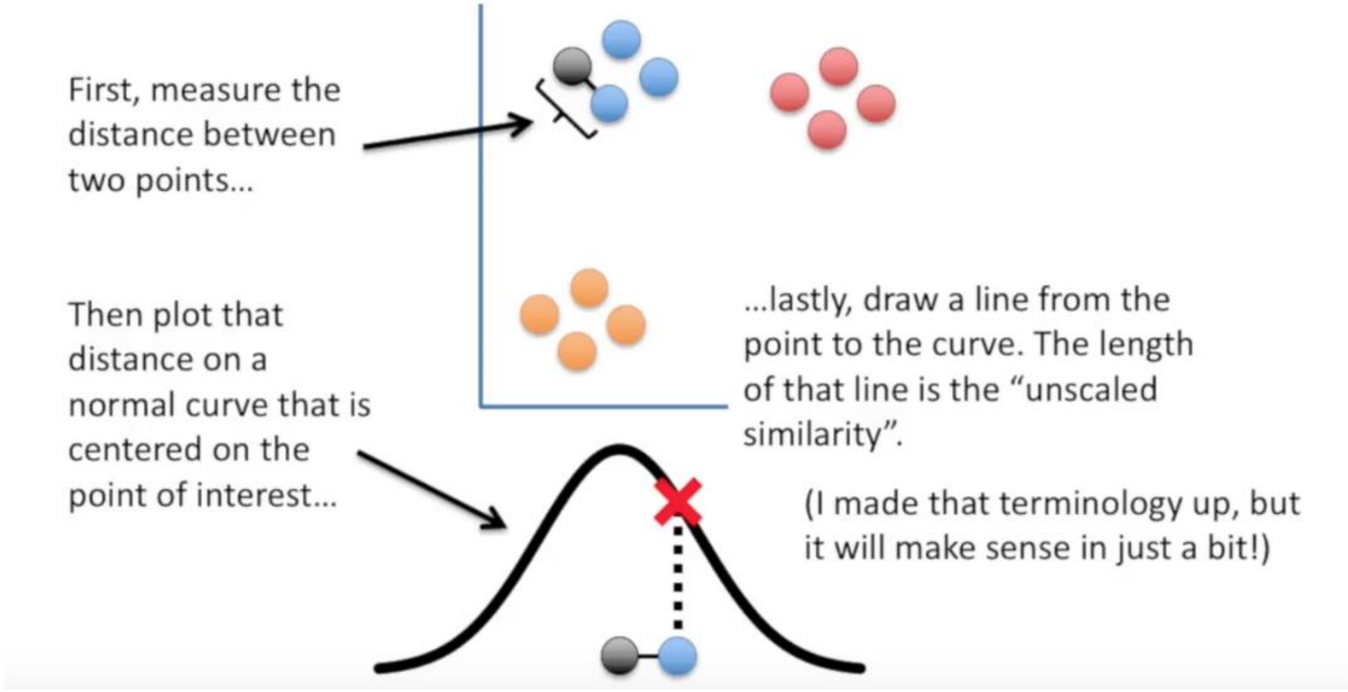
시각화 결과 :


Breast Cacner Dataset MDS (n_components = 2)
malignant
benign
Second MDS dimension
First MDS dimension
15
10
5
0
−5
−10
−20
−15
−10
−5
0
5
10

#03 T-SNE

T-SNE란?

- Manifold Learning Algorithm의 일종 : 고차원 데이터를 2차원 데이터로 축소, 시각화에 주로 사용
- 비슷한 샘플은 가까이, 비슷하지 않은 샘플은 멀리 떨어지도록 하면서 차원을 축소



#03 T-SNE

사이킷런 : TSNE를 이용해 구현 가능

```python
from sklearn.manifold import TSNE

tsne = TSNE(random_state=42)

canc_tsne = tsne.fit_transform(canc_norm)

print('Number of Features in Breast Cancer Dataset Before T-SNE : {}\n\nNumber
 of Featrues in Breast Cancer Dataset After T-SNE: {}'.format(X_canc.shape[1],
 canc_tsne.shape[1]))

plot_labelled_scatter(canc_tsne, y_canc, ['malignant', 'benign'], (15, 9))

plt.xlabel('First T-SNE Dimension', fontsize=14)
plt.ylabel('Second T-SNE Dimension', fontsize=14)
plt.title('Breast Cancer Dataset T-SNE', fontsize=17)
```

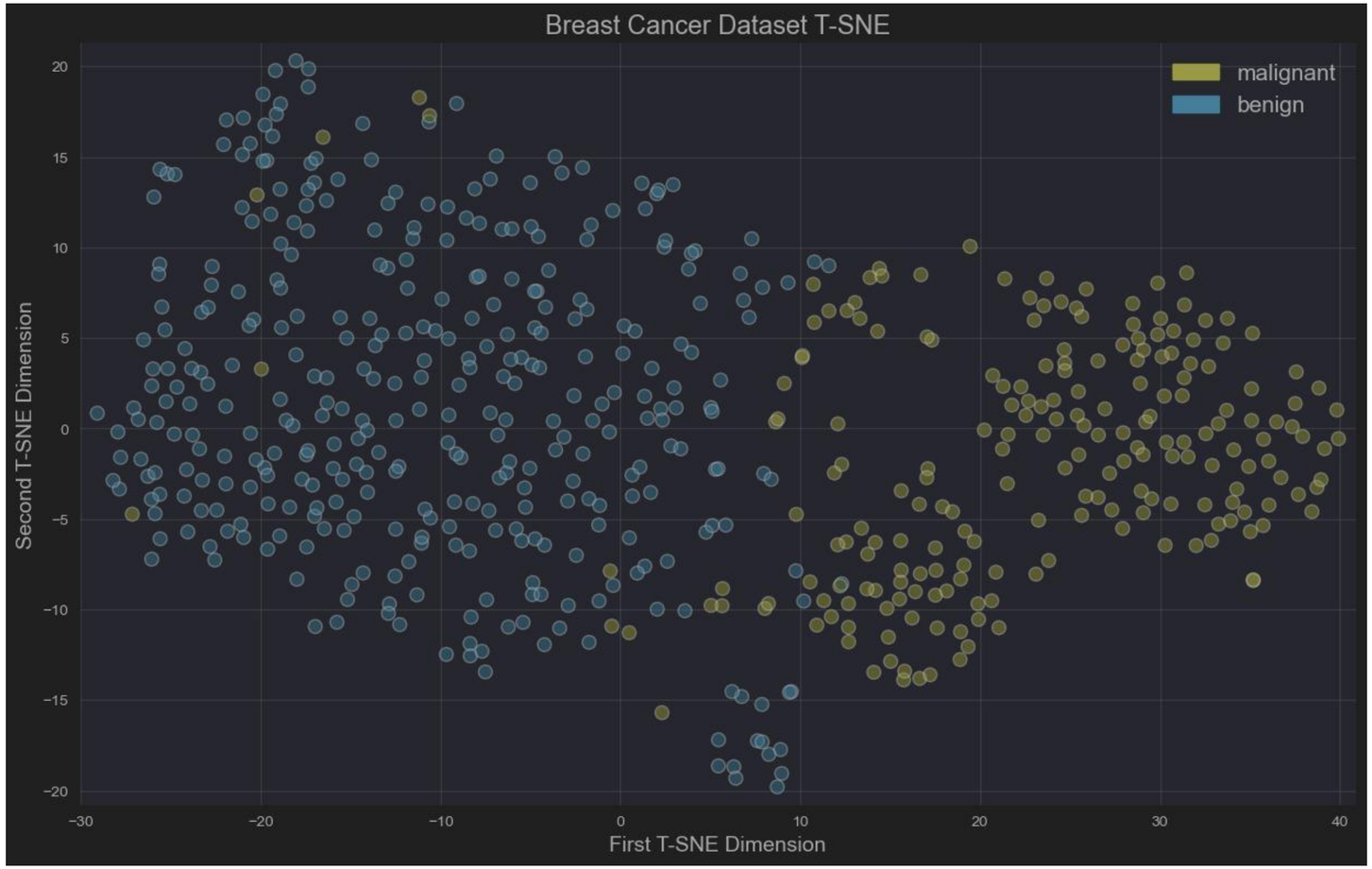
Executed at 2024.05.13 00:15:59 in 4s 585ms

```
 Number of Features in Breast Cancer Dataset Before T-SNE : 30

 Number of Featrues in Breast Cancer Dataset After T-SNE: 2
```

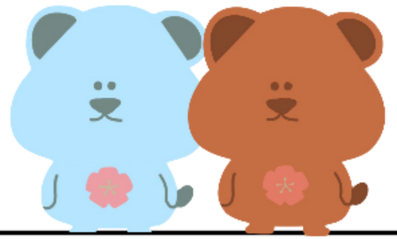
차원 축소 후 피처 수 감소 확인 가능!

#03 T-SNE

시각화 결과 :



THANK YOU



EWHA EURON