06_비지도 학습

비지도학습: y데이터 제외하고 다른 변수들 학습/ 정답 없음

차원축소/군집

군집과 분류는 Y값 유무의 차이만 있음

비지도 학습

• 차원 축소

차원: 피처의 갯수 -1개 1차원, 2개 2차원...

02

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

> 차원 축소를 사용하여 피처 수를 줄이면 더 직관적으로 데이터를 해석할 수 있다.

> 일반적으로 피처 선택과 피처 추출로 나눌 수 있다.

- 피처 선택 : 불필요한 피처를 제거하고 특징을 잘 나타내는 주요 피처만 선택
- 피처 추출 : 기존 피처를 압축해서 추출하는 것 : 차원 축소

> 대표적인 알고리즘으로는 PCA, LDA, SVD, NMF 등이 있다.

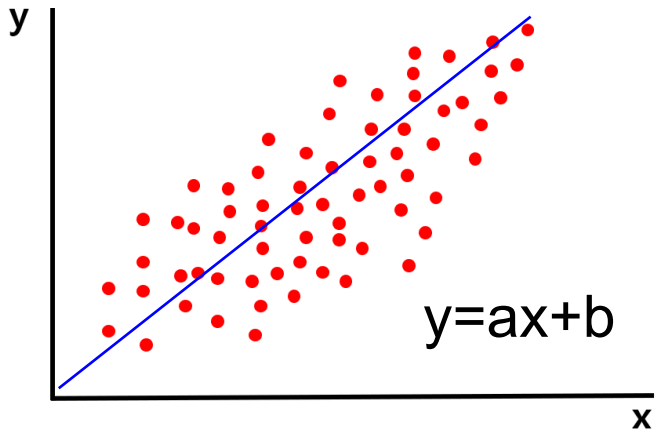
단점: 어떻게 압축되었는지 잘 알 수 없음
지워진 정보를 알 수 없음

장점: 시간,비용 절약/효과가 더 좋을 수도

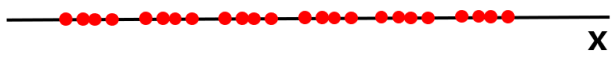
• PCA(Principal Component Analysis) 주성분 분석

> 가장 대표적인 차원 축소 기법

> 여러 변수 간에 상관관계를 이용해 주성분을 추출해 차원을 축소하는 기법



<2차원 데이터>

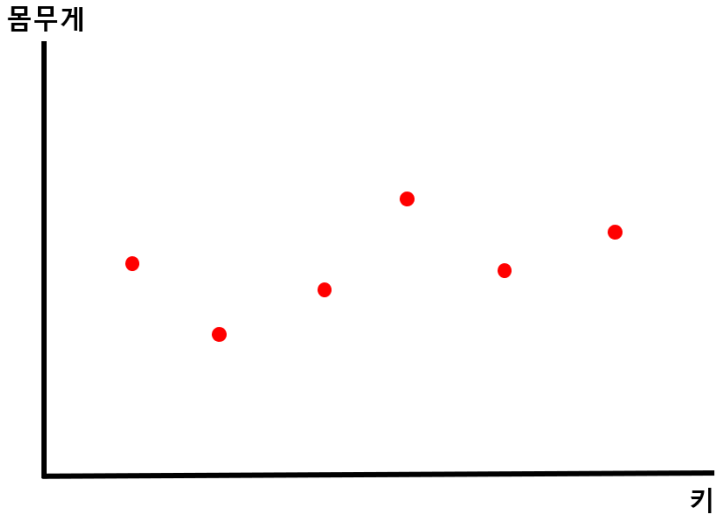


<1차원 데이터>

• PCA(Principal Component Analysis)

> 직관적 해석
  - 키와 몸무게를 2차원 그래프로 나타냄

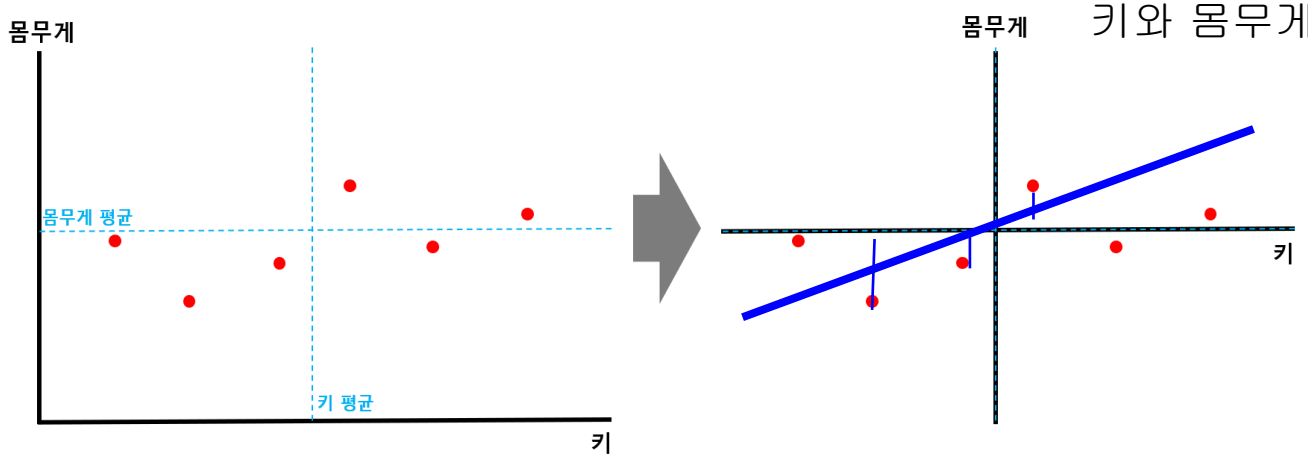
	키	몸무게
사람 1	170	68
사람 2	174	72
사람 3	172	84
사람 4	176	76
사람 5	168	60
사람 6	166	74



• PCA(Principal Component Analysis)

> 직관적 해석

- 평균을 원점으로 하는 그래프로 나타냄

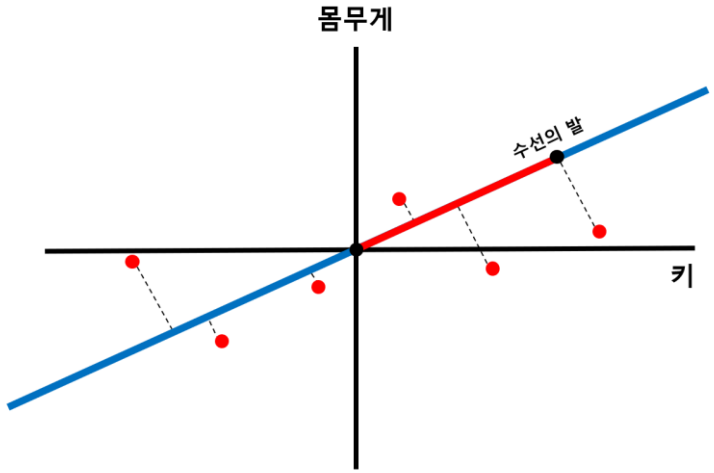


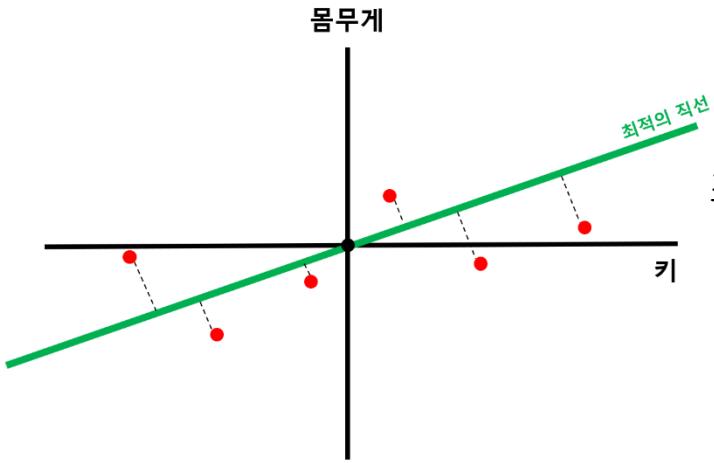
키와 몸무게 정보를 가진 대각선 생성

비지도 학습

- PCA(Principal Component Analysis)

> **직관적 해석**
  - 평균을 원점으로 하는 그래프로 나타냄 :평균은 꼭 지나가야한다는 점에서 회귀선과 다름
  - 오차의 제곱의 합이 가장 작은 최적의 직선을 구함

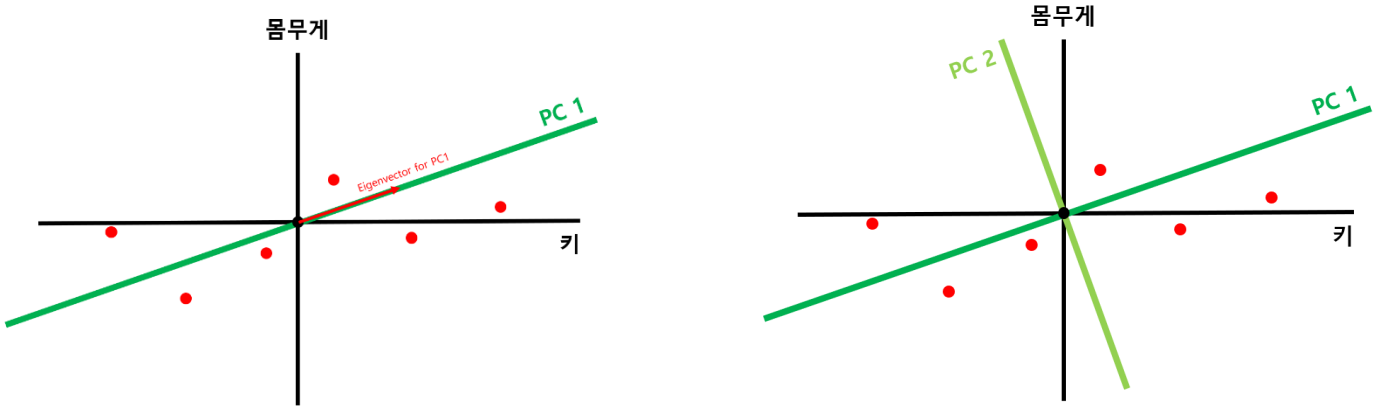




최적의 직선

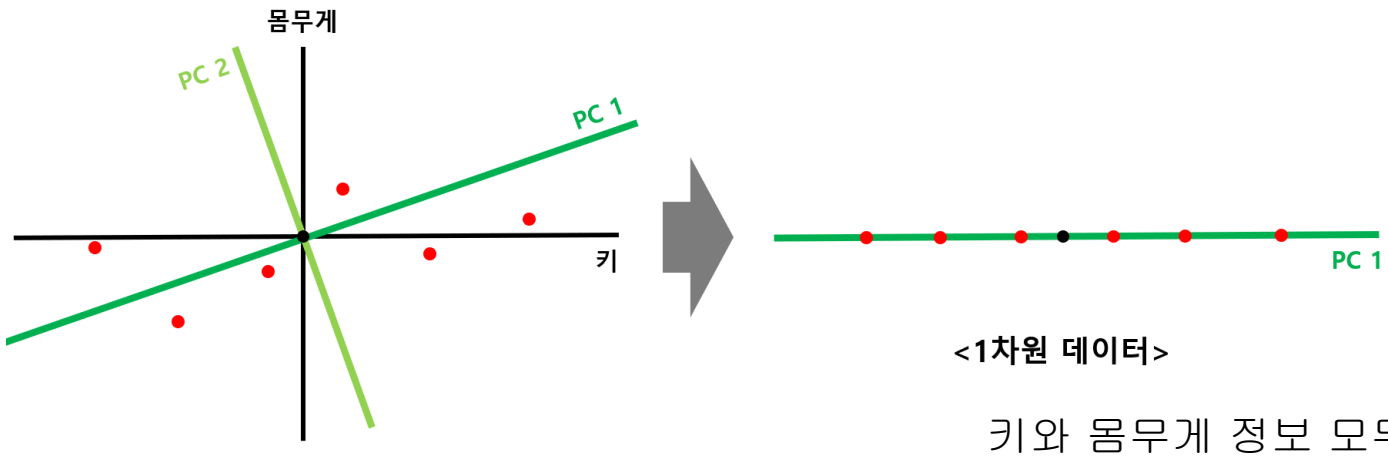
• PCA(Principal Component Analysis)

> **직관적 해석**
  - 최적의 선을 PC1이라 잡는다.
  - 원점을 지나는 수직의 PC2 선으로 잡는다(N개의 피처에서 PC선은 N가 나온다)



• PCA(Principal Component Analysis)

> **직관적 해석**
  - PC1만 사용하면 2차원 데이터를 1차원 데이터로 바꿀 수 있다.



<2차원 데이터>

<1차원 데이터>

키와 몸무게 정보 모두 가짐

• PCA(Principal Component Analysis)

> 아이리스 데이터 PCA 실습

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

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

• PCA(Principal Component Analysis)

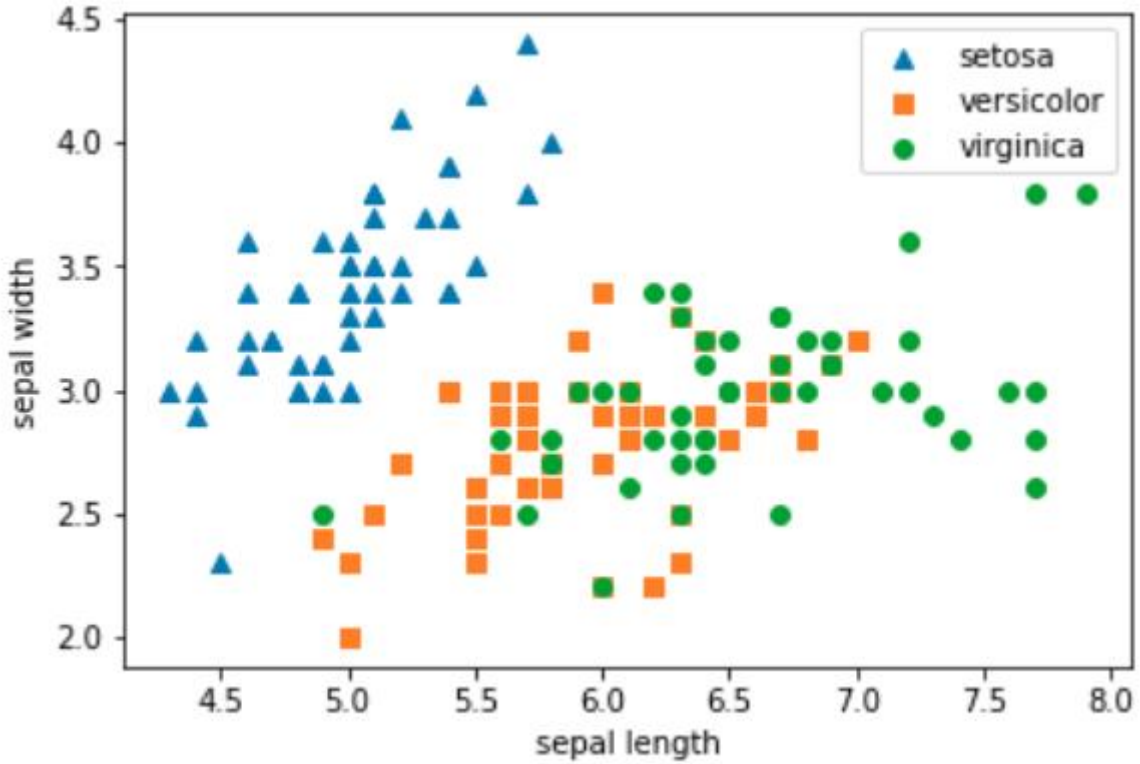
> 아이리스 데이터 PCA 실습

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

- PCA(Principal Component Analysis)

> 아이리스 데이터 PCA 실습



• PCA(Principal Component Analysis)

> 아이리스 데이터 PCA 실습

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

scaler = StandardScaler()
iris_scaled = scaler.fit_transform(irisDF.iloc[:,:-1])

pca = PCA(n_components = 2) 갯수를 쓰지않으면 원래 값으로
pca.fit(iris_scaled)
iris_pca = pca.transform(iris_scaled)
```

1. X,y 안나누고
2. 예측 없고
3. 꼭 스케일링 함: 정규화

• PCA(Principal Component Analysis)

> 아이리스 데이터 PCA 실습

```
pca_columns = ['PC1', 'PC2']
irisDF_pca = pd.DataFrame(iris_pca, columns = pca_columns)
irisDF_pca['target'] = iris.target
irisDF_pca.head()
```

Out :

	PC1	PC2	target
0	-2.264703	0.480027	0
1	-2.080961	-0.674134	0
2	-2.364229	-0.341908	0
3	-2.299384	-0.597395	0
4	-2.389842	0.646835	0

n_compoment가 2이기에 2개의 PC선
만약 3이었다면 3개 ==> 4차원 데이터이므로

• PCA(Principal Component Analysis)

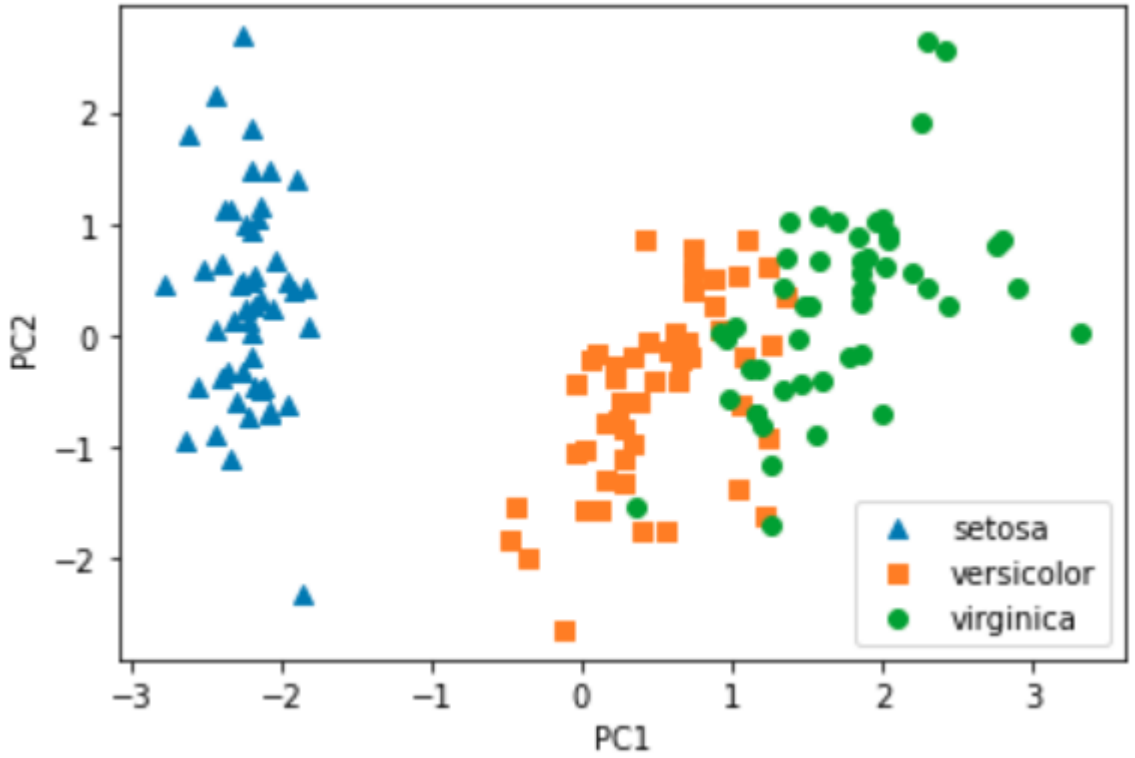
> 아이리스 데이터 PCA 실습

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

for i, marker in enumerate(markers):
    x_axis_data = irisDF_pca[irisDF_pca['target'] == i]['PC1']
    y_axis_data = irisDF_pca[irisDF_pca['target'] == i]['PC2']
    plt.scatter(x_axis_data, y_axis_data, marker = marker, label = iris.target_names[i])
plt.legend()
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.show()
```

- PCA(Principal Component Analysis)

> 아이리스 데이터 PCA 실습



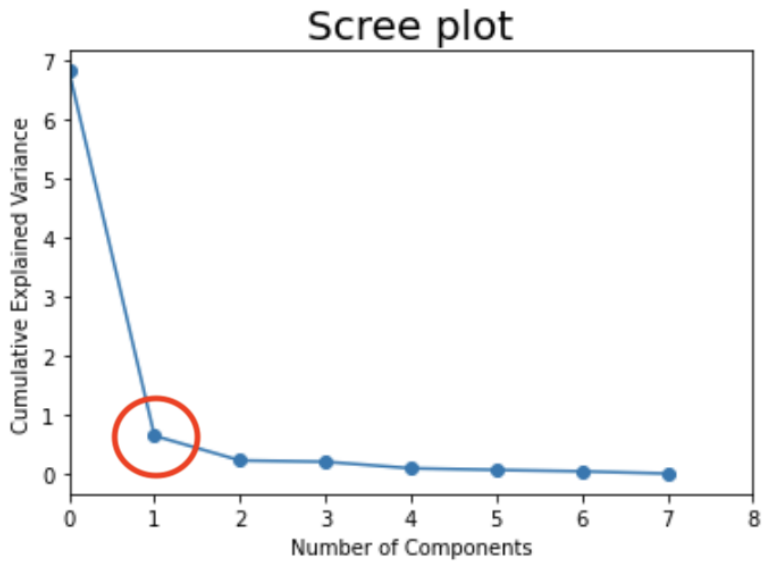
• PCA(Principal Component Analysis)

> 차원 축소의 최적의 개수 알아내기

- Scree Plot을 그려서 확인하기

```
# PCA 개별 변동성 비율
pca.explained_variance_ratio_
```

Out : [0.72962445, 0.22850762]



• PCA(Principal Component Analysis)

> Scree Plot 그리기

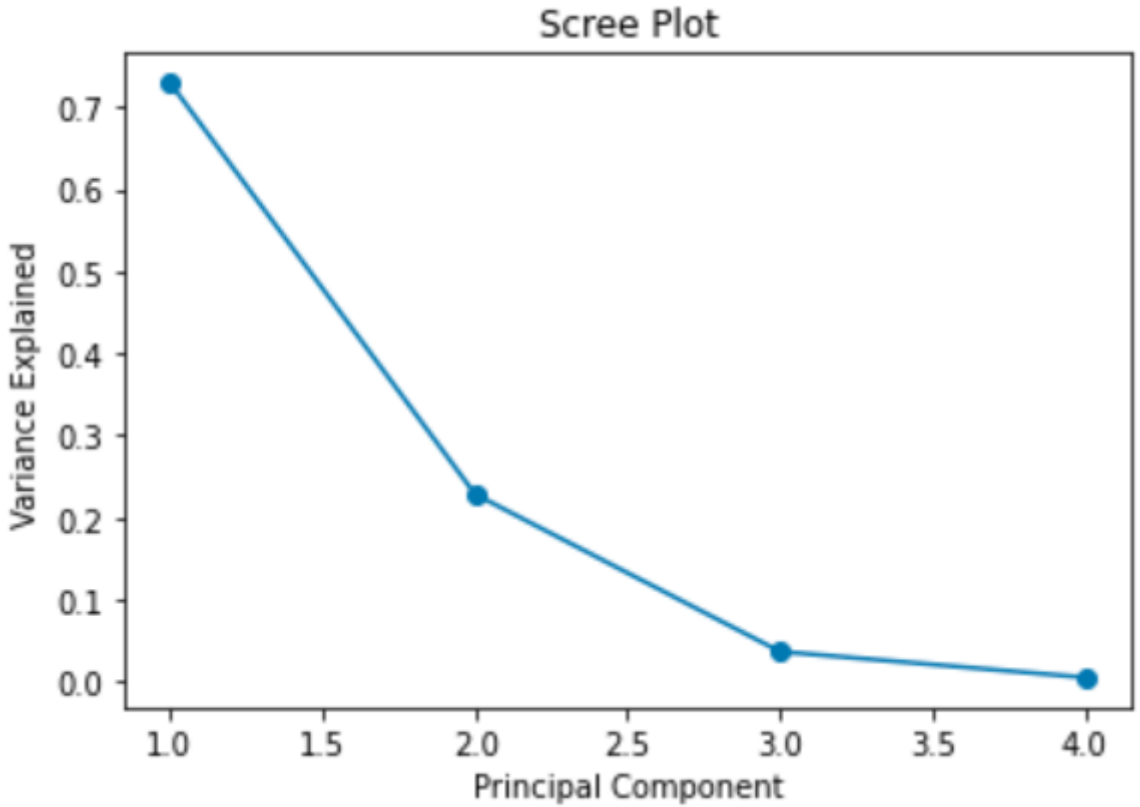
```
import numpy as np

pca = PCA(n_components = 4) # 피처 개수만큼
pca.fit(iris_scaled)
iris_pca = pca.transform(iris_scaled)

PC_values = np.arange(pca.n_components_) + 1
plt.plot(PC_values, pca.explained_variance_ratio_, 'o-')
plt.title('Scree Plot')
plt.xlabel('Principal Component')
plt.ylabel('Variance Explained')
plt.show()
```

- PCA(Principal Component Analysis)

> Scree Plot 그리기



기본적으로 누적 비율이 80% 이상인 주성분까지 추출

• PCA(Principal Component Analysis)

> 분류로 정확도 비교하기

```
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('원본 데이터 교차 검증 개별 정확도:',scores)
print('원본 데이터 평균 정확도:', np.mean(scores))
```

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

• PCA(Principal Component Analysis)

> 분류로 정확도 비교하기

```
pca_X = irisDF_pca[['PC1', 'PC2']]
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))
```

Out : PCA 변환 데이터 교차 검증 개별 정확도: [0.88 0.88 0.88]

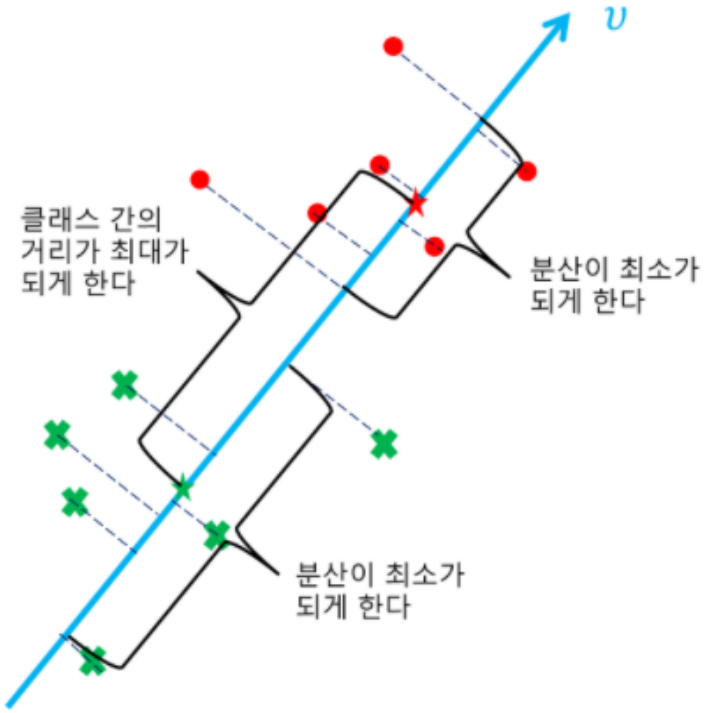
PCA 변환 데이터 평균 정확도: 0.88

• LDA(Linear Discriminant Analysis)

> 선형 판별 분석법으로 PCA와 유사

> 개별 클래스를 분별할 수 있는 기준을 유지하면서 차원 축소

> 클래스 간의 거리는 멀리하고 클래스 내 분산이 최소가 되게한다.



범주형 변수 분석에 효과적
: 지도 학습으로 볼 수 있음
종속변수 있음

• LDA(Linear Discriminant Analysis)

> 아이리스 데이터 LDA 실습

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

iris = load_iris()
scaler = StandardScaler()
iris_scaled = scaler.fit_transform(iris.data)
lda = LinearDiscriminantAnalysis(n_components=2)
lda.fit(iris_scaled, iris.target)
iris_lda = lda.transform(iris_scaled)
print(lda.explained_variance_ratio_)
```

PCA와 다르게 target 값을 같이 넣어준다.

• LDA(Linear Discriminant Analysis)

> 아이리스 데이터 LDA 실습

Out : [0.9912126 0.0087874]

```
lda_columns=['lda1','lda2']
irisDF_lda = pd.DataFrame(iris_lda, columns = lda_columns)
irisDF_lda['target']=iris.target
irisDF_lda.head()
```

Out :

	lda1	lda2	target
0	8.061800	0.300421	0
1	7.128688	-0.786660	0
2	7.489828	-0.265384	0
3	6.813201	-0.670631	0
4	8.132309	0.514463	0

PCA보다 집단간에는 더 떨어져 있음
집단내에는 더 가까움

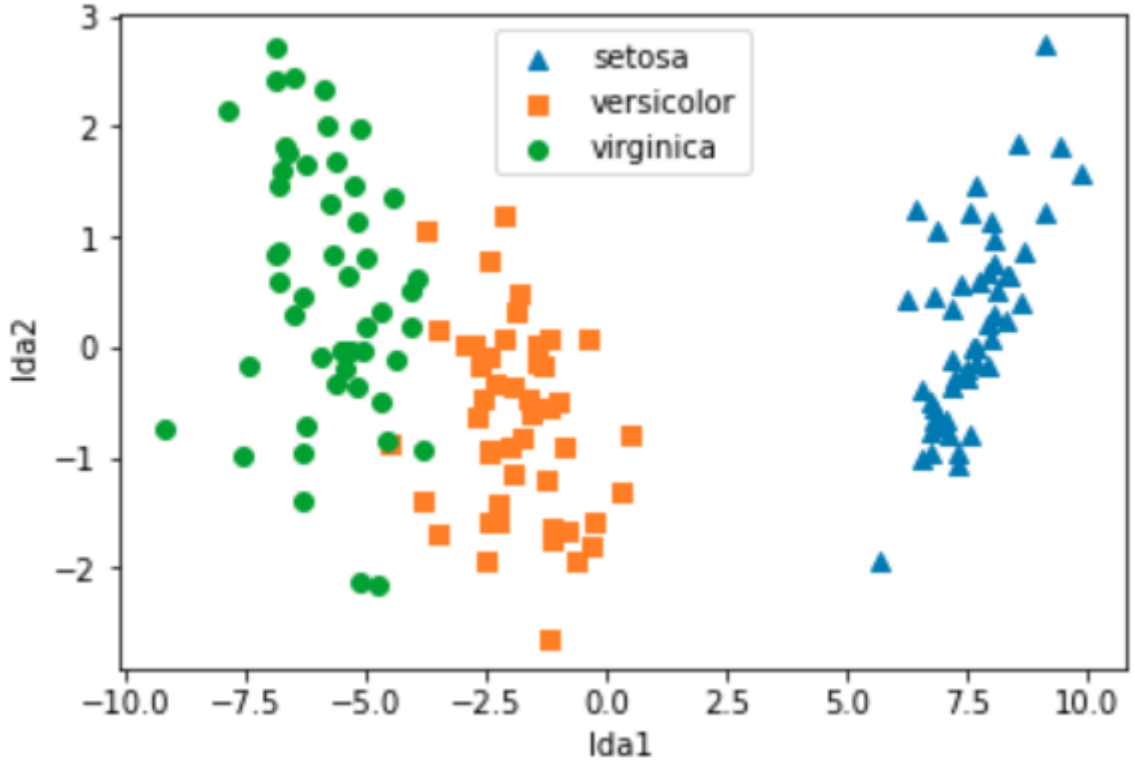
• LDA(Linear Discriminant Analysis)

> 아이리스 데이터 LDA 실습

```
markers=['^', 's', 'o']
for i, marker in enumerate(markers):
    x_axis_data = irisDF_lda[irisDF_lda['target']==i]['lda1']
    y_axis_data = irisDF_lda[irisDF_lda['target']==i]['lda2']
    plt.scatter(x_axis_data, y_axis_data, marker=marker, label=iris.target_names[i])
plt.legend()
plt.xlabel('lda1')
plt.ylabel('lda2')
plt.show()
```

- LDA(Linear Discriminant Analysis)

> 아이리스 데이터 LDA 실습



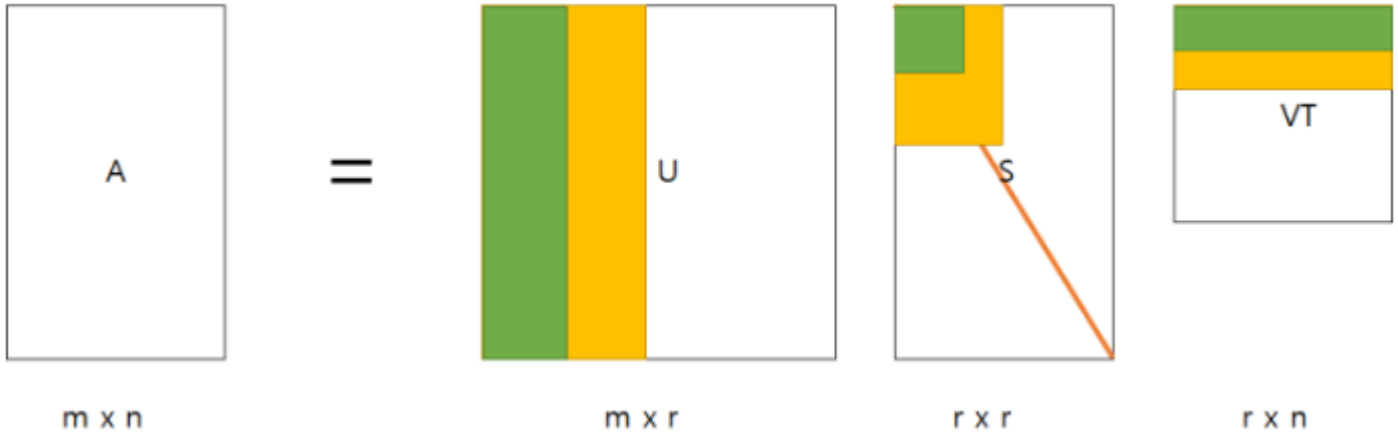
• SVD(Singular Value Decomposition)

PCA의 원조 제거와 추출을 동시에 이루어저짐

> 행렬의 곱을 통해 벡터를 선형으로 변환시키는 방법

> 다중 공선성이 높은 값은 제거되어 피처 추출이 진행된다.

> 그 중에서도 상위 몇 개만 선택하여 차원 축소를 진행한다.



• SVD(Singular Value Decomposition)

> 아이리스 데이터 SVD 실습

```
from sklearn.decomposition import TruncatedSVD

                                                    PCA보다 성능 조금 더 좋음
tsvd = TruncatedSVD(n_components = 2)
tsvd.fit(iris_scaled)
iris_tsvd = tsvd.transform(iris_scaled)
tsvd_columns=['tsvd1','tsvd2']
irisDF_tsvd = pd.DataFrame(iris_tsvd, columns = tsvd_columns)
irisDF_tsvd['target']=iris.target
irisDF_tsvd.head()
```

• SVD(Singular Value Decomposition)

> 아이리스 데이터 SVD 실습

Out :

	tsvd1	tsvd2	target
0	-2.264703	0.480027	0
1	-2.080961	-0.674134	0
2	-2.364229	-0.341908	0
3	-2.299384	-0.597395	0
4	-2.389842	0.646835	0

```
tsvd.explained_variance_ratio_
```

Out : array([0.72962445, 0.22850762])

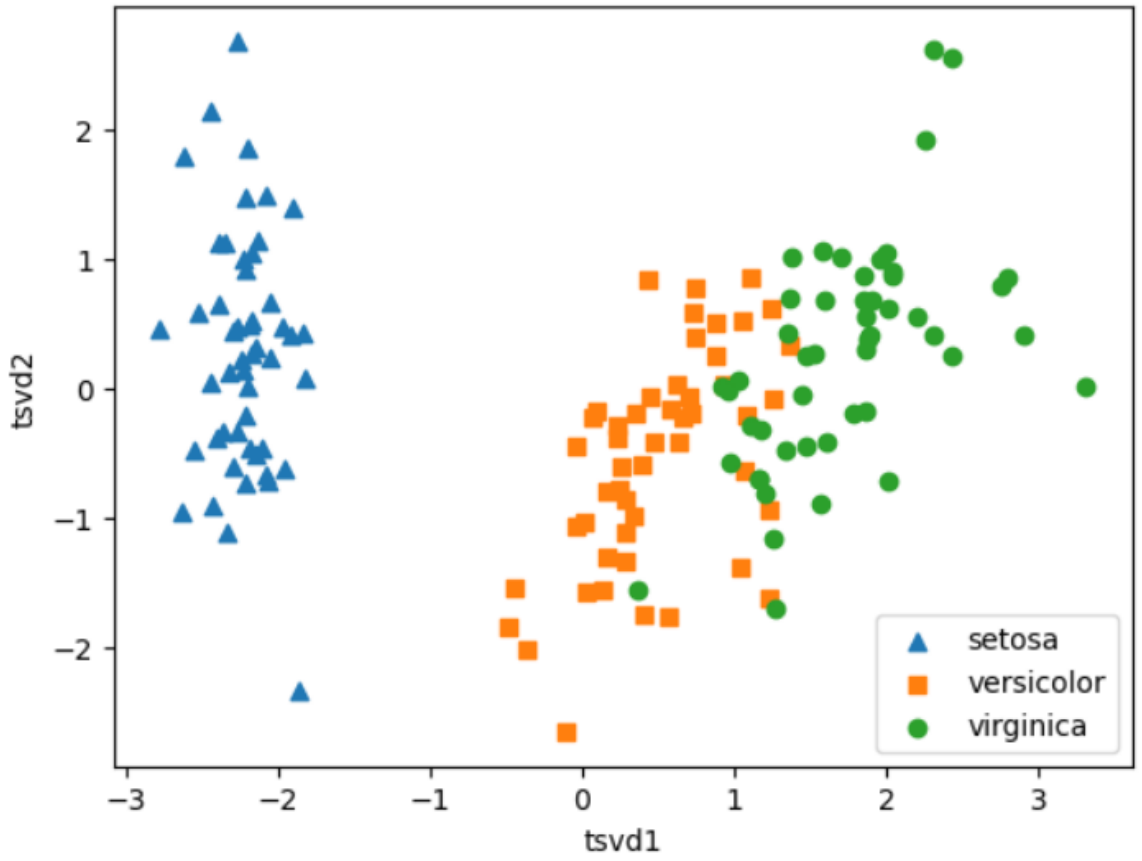
• SVD(Singular Value Decomposition)

> 아이리스 데이터 SVD 실습

```
markers=['^', 's', 'o']
for i, marker in enumerate(markers):
    x_axis_data = irisDF_tsvd[irisDF_tsvd['target']==i]['tsvd1']
    y_axis_data = irisDF_tsvd[irisDF_tsvd['target']==i]['tsvd2']
    plt.scatter(x_axis_data, y_axis_data, marker=marker, label=iris.target_names[i])
plt.legend()
plt.xlabel('tsvd1')
plt.ylabel('tsvd2')
plt.show()
```

• SVD(Singular Value Decomposition)

> 아이리스 데이터 SVD 실습



• NMF(Non-Negative Matrix Factorization)

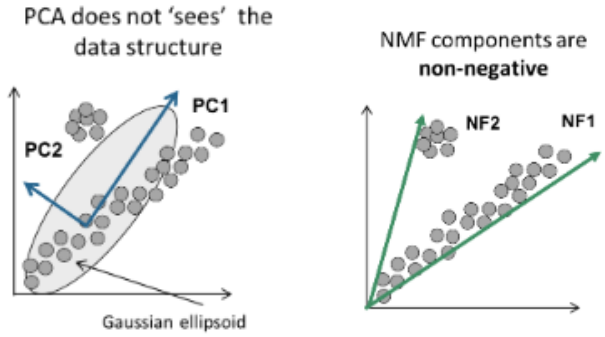
> 음수를 포함하지 않는 행렬 분해

$$\begin{pmatrix} A & A & A & A & A & A \\ A & A & A & A & A & A \\ A & A & A & A & A & A \\ A & A & A & A & A & A \end{pmatrix} = \begin{pmatrix} W & W \\ W & W \\ W & W \\ W & W \end{pmatrix} \begin{pmatrix} H & H & H & H & H & H \\ H & H & H & H & H & H \end{pmatrix}$$

데이터 행렬분해

> 데이터를 W와 H 행렬의 곱으로 나타낼 수 있고 이때 W에 잠재요소를 사용

> 양수에 있어서 PCA나 SVD보다 더 잘 피처 추출을 한다고 한다.



• NMF(Non-Negative Matrix Factorization)

> 아이리스 데이터 NMF 실습

```
from sklearn.decomposition import NMF


nmf = NMF(n_components= 2)
nmf.fit(iris.data)
iris_nmf = nmf.transform(iris.data)
nmf_columns=['nmf1','nmf2']
irisDF_nmf = pd.DataFrame(iris_nmf, columns = nmf_columns)
irisDF_nmf['target']=iris.target
irisDF_nmf.head()
```

음수가 들어가면 안되므로 스케일링 하지않은 데이터를 입력한다.

• NMF(Non-Negative Matrix Factorization)

> 아이리스 데이터 NMF 실습

Out :

	nmf1	nmf2	target
0	0.413500	0.104673	0
1	0.365449	0.140978	0
2	0.377800	0.101885	0
3	0.349981	0.148964	0
4	0.415896	0.095306	0

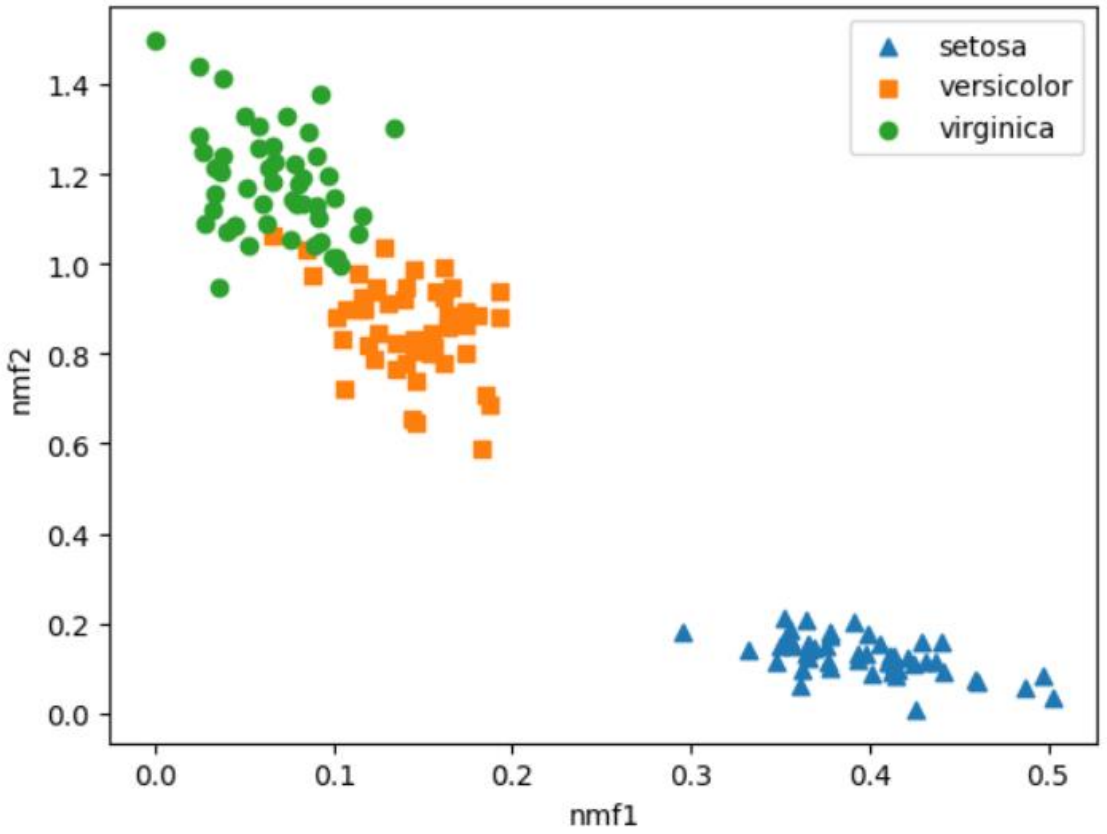
• NMF(Non-Negative Matrix Factorization)

> 아이리스 데이터 NMF 실습

```
markers=['^', 's', 'o']
for i, marker in enumerate(markers):
    x_axis_data = irisDF_nmf[irisDF_nmf['target']==i]['nmf1']
    y_axis_data = irisDF_nmf[irisDF_nmf['target']==i]['nmf2']
    plt.scatter(x_axis_data, y_axis_data, marker=marker, label=iris.target_names[i])
plt.legend()
plt.xlabel('nmf1')
plt.ylabel('nmf2')
plt.show()
```

- NMF(Non-Negative Matrix Factorization)

> 아이리스 데이터 NMF 실습

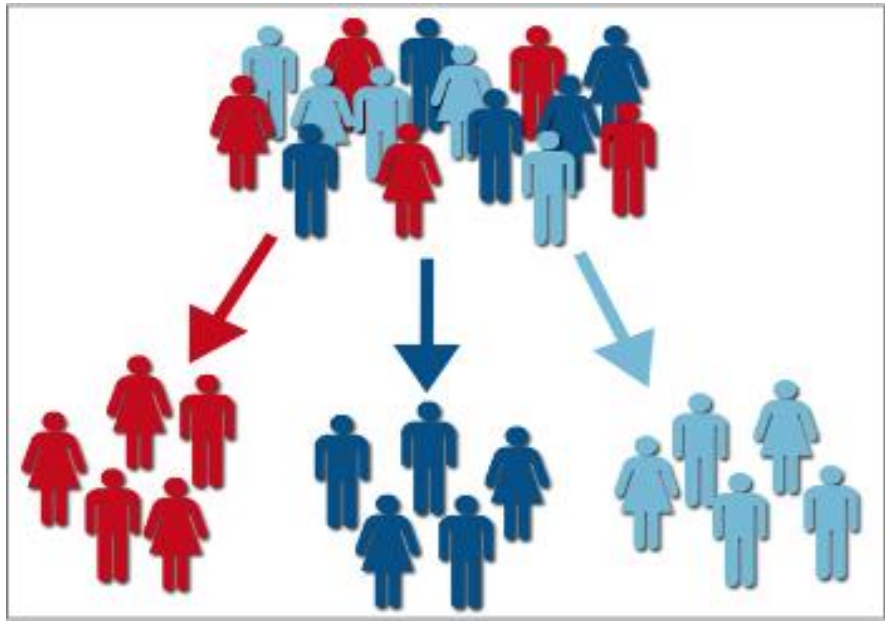


• 군집화(Clustering)

> 비지도 학습의 대표적인 기술로 데이터를 그룹핑 하는 방법

> 레이블(종속변수)가 없는 상황에서 피처의 특성만으로 분류한다.

> 대표적인 알고리즘으로 K-평균, DBSCAN 등이 있다.

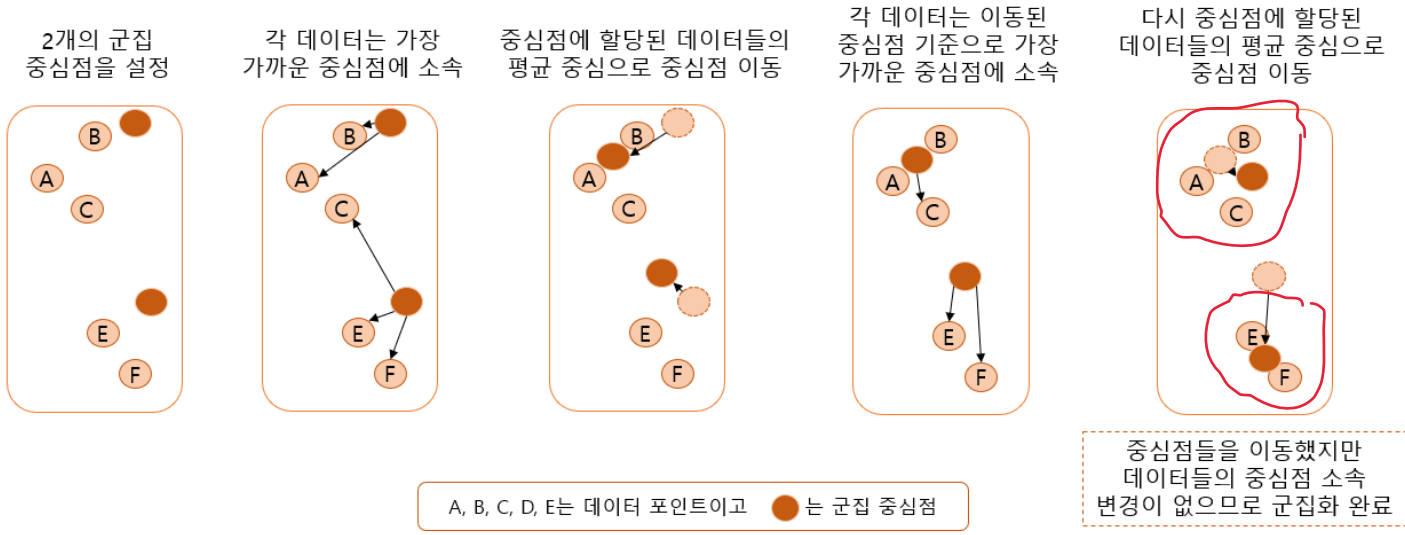


• K-평균 군집화(K-means clustering)

k개의 평균을 중심으로 군집

> 가장 일반적으로 사용되는 군집 알고리즘

> 군집 중심점을 선택해 해당 중심에서 가장 가까운 포인트를 선택하는 기법



새로운 군집으로 묶임

더이상 이동이 없으면 자동 멈춤
ma_iter 횟수 내에서

• K-평균 군집화(K-means clustering)

> 아이리스 데이터 세트를 사용한 k-평균 군집화 실습

```
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

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

비지도 학습

• K-평균 군집화(K-means clustering)

> 아이리스 데이터 세트를 사용한 k-평균 군집화 실습

```
kmeans = KMeans(n_clusters=3, max_iter=300, random_state=0)
kmeans.fit(irisDF)
print(kmeans.labels_)
```

Out :

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 2 2 2 0 2 2 2 2
 2 2 0 0 2 2 2 2 0 2 0 2 0 2 2 0 0 2 2 2 2 2 0 2 2 2 2 0 2 2 2 0 2 2 2 0 2
 2 0]
```

• K-평균 군집화(K-means clustering)

> 아이리스 데이터 세트를 사용한 k-평균 군집화 실습

```
irisDF['target'] = iris.target
irisDF['cluster']=kmeans.labels_
iris_result = irisDF.groupby(['target', 'cluster'])['sepal_length'].count()
print(iris_result)
```

Out :

```
target  cluster
0       1          50
1       0          48
        2           2
2       0          14
        2          36
Name: sepal_length, dtype: int64
```

에측 확인하니 오차 좀 많음
: 거리로 분석하다보니.. 한계있음

• K-평균 군집화(K-means clustering)

> 아이리스 데이터 세트를 사용한 k-평균 군집화 실습

```
from sklearn.decomposition import PCA

scaler = StandardScaler()
scaled = scaler.fit_transform(irisDF.iloc[:,:4])
pca = PCA(n_components=2)
pca_transformed = pca.fit_transform(scaled)

irisDF['PC1'] = pca_transformed[:,0]
irisDF['PC2'] = pca_transformed[:,1]
```

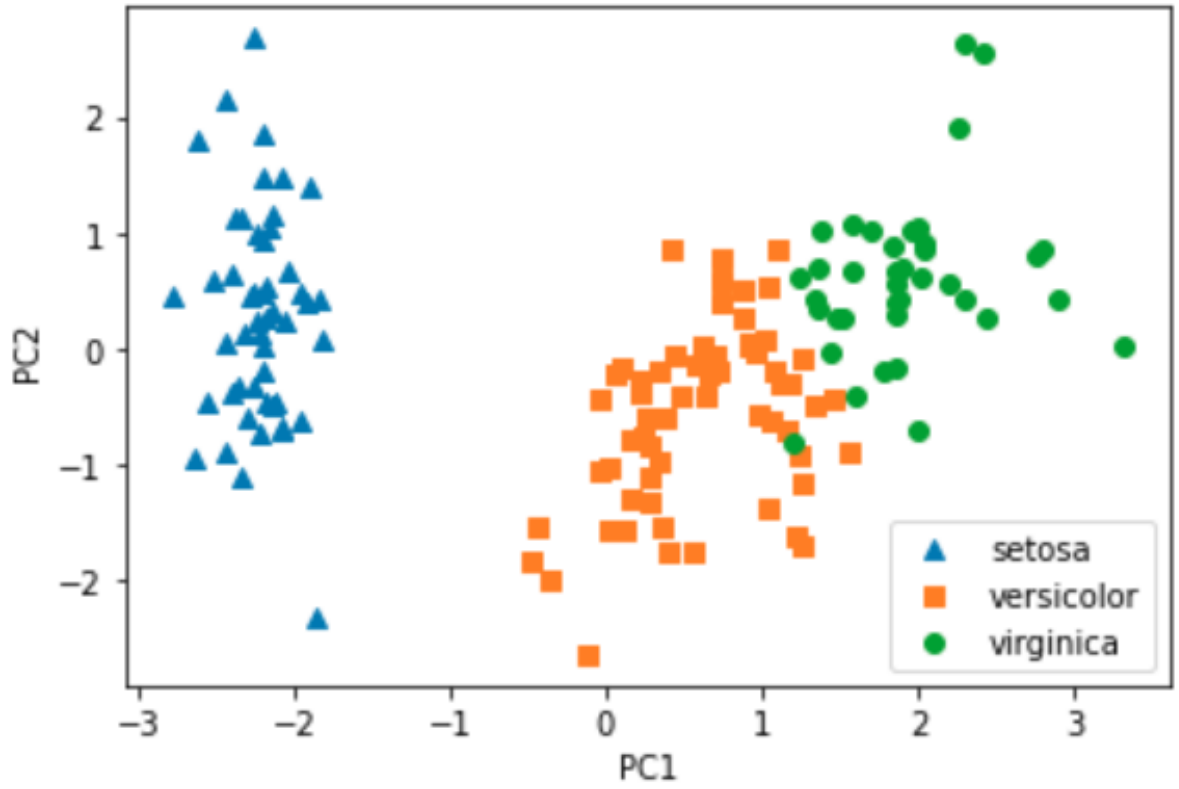
• K-평균 군집화(K-means clustering)

> 아이리스 데이터 세트를 사용한 k-평균 군집화 실습

```
markers = ['^', 's', 'o']
for i, marker in enumerate(markers):
    x_axis_data = irisDF[irisDF['cluster'] == i]['PC1']
    y_axis_data = irisDF[irisDF['cluster'] == i]['PC2']
    plt.scatter(x_axis_data, y_axis_data, marker = marker, label = iris.target_names[i])
plt.legend()
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.show()
```

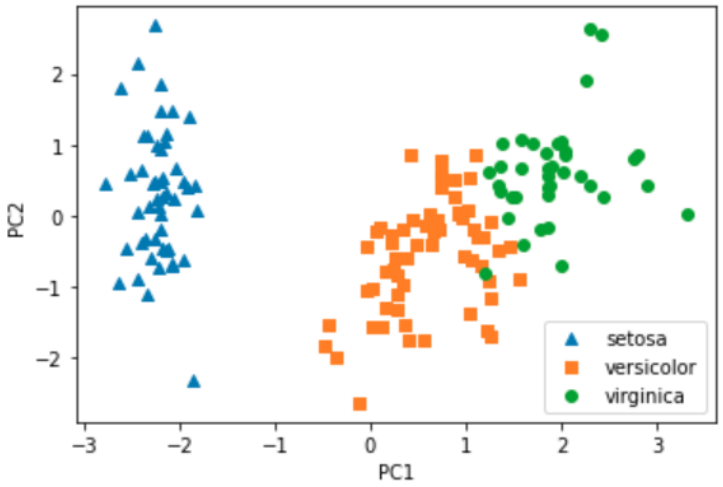
- K-평균 군집화(K-means clustering)

> 아이리스 데이터 세트를 사용한 k-평균 군집화 실습

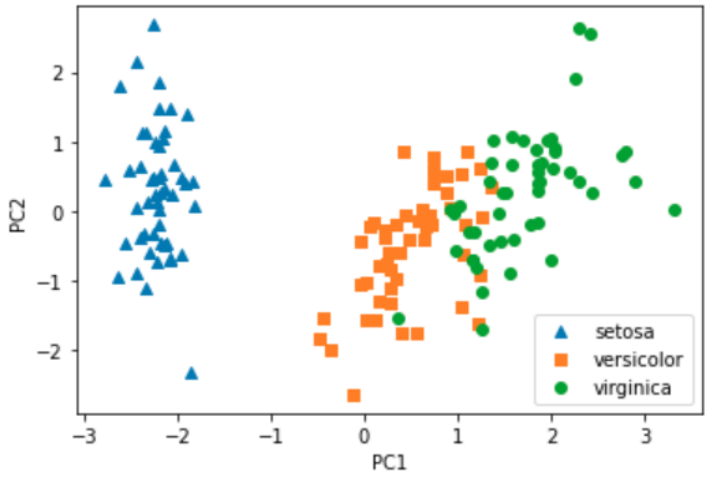


• K-평균 군집화(K-means clustering)

> 아이리스 데이터 세트를 사용한 k-평균 군집화 실습



< 군집화 결과>



<원본 레이블 >

• K-평균 군집화(K-means clustering)

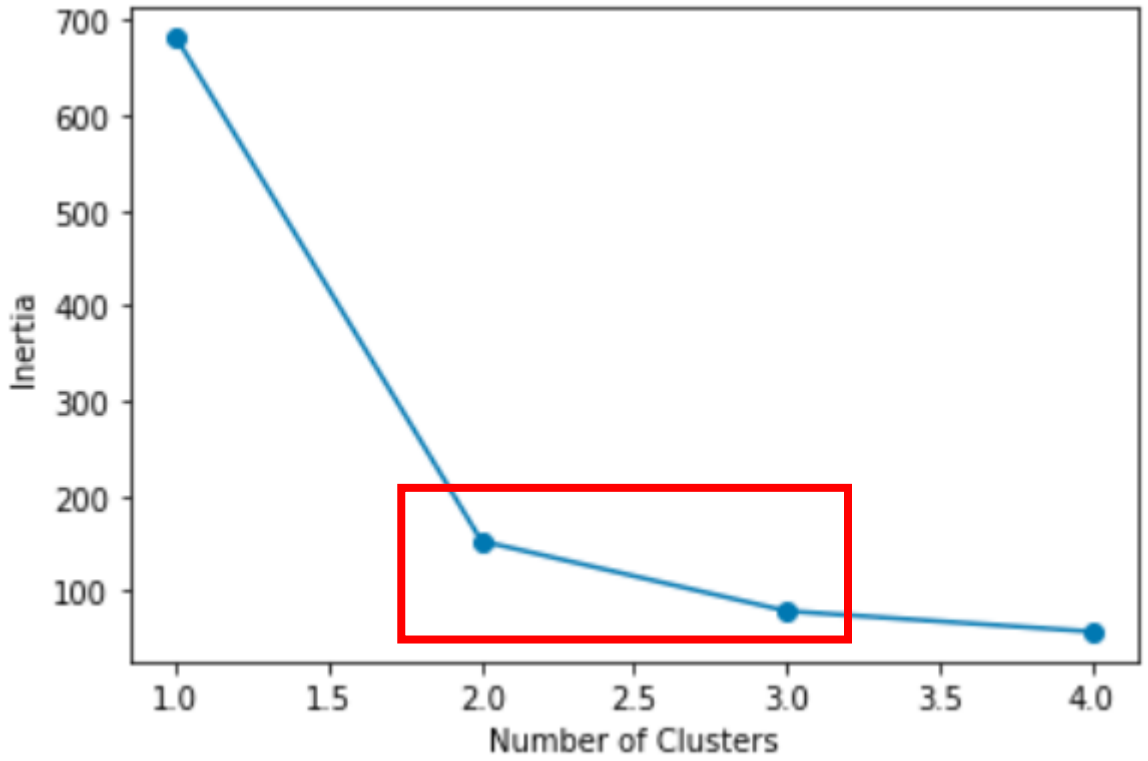
> 최적의 군집 개수 찾기

```
inertia = []
for i in range(1,5):
    kmeans_plus = KMeans(n_clusters = i, max_iter=300, random_state=0)
    kmeans_plus.fit(irisDF.iloc[:,:4])
    inertia.append(kmeans_plus.inertia_)

plt.plot(range(1,5), inertia, 'o-')
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.show()
```

• K-평균 군집화(K-means clustering)

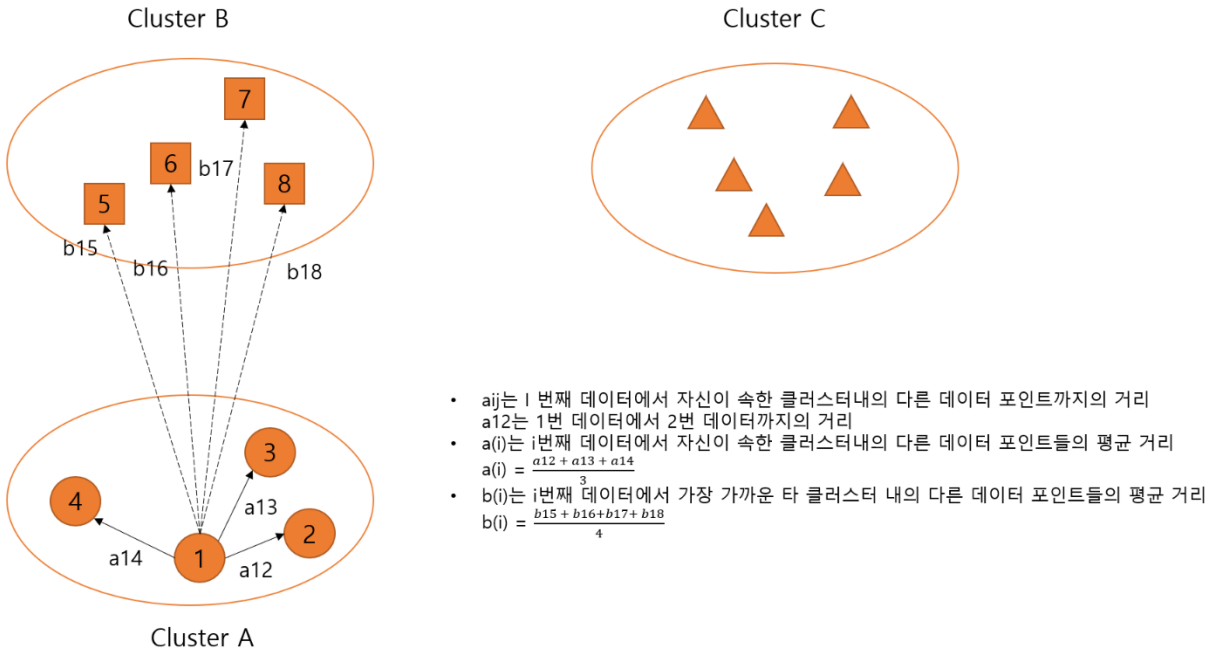
> 최적의 군집 개수 찾기



횟수를 더할때 가장 변화량이 큰 횟수를 선택한다,: 여긴 2회 또는 3화

• 군집 평가(Cluster Evaluation)

> 실루엣 분석을 사용한 군집 평가
  - 얼마나 효율적으로 분리돼 있는지를 확인



- aij는 I 번째 데이터에서 자신이 속한 클러스터내의 다른 데이터 포인트까지의 거리
  a12는 1번 데이터에서 2번 데이터까지의 거리
- a(i)는 i번째 데이터에서 자신이 속한 클러스터내의 다른 데이터 포인트들의 평균 거리
  $a(i) = \frac{a12 + a13 + a14}{3}$
- b(i)는 i번째 데이터에서 가장 가까운 타 클러스터 내의 다른 데이터 포인트들의 평균 거리
  $b(i) = \frac{b15 + b16 + b17 + b18}{4}$

  - -1 ~ 1 사이의 값을 가지며 -1에 가까울수록 좋지 않고 1에 가까울수록 좋다.

• 군집 평가(Cluster Evaluation)

> 실루엣 분석 실습

```
from sklearn.metrics import silhouette_samples, silhouette_score

score_samples = silhouette_samples(iris.data, irisDF['cluster'])
print('silhouette_samples( ) return 값의 shape' , score_samples.shape)

irisDF['silhouette_coeff'] = score_samples
average_score = silhouette_score(iris.data, irisDF['cluster'])
print('붓꽃 데이터셋 Silhouette Analysis Score:{0:.3f}'.format(average_score))
irisDF.head()
```

• 군집 평가(Cluster Evaluation)

> 실루엣 분석 실습

Out : silhouette_samples( ) return 값의 shape (150,)

붓꽃 데이터셋 Silhouette Analysis Score:0.553

	sepal_length	sepal_width	petal_length	petal_width	target	cluster	PC1	PC2	silhouette_coeff
0	5.1	3.5	1.4	0.2	0	1	-2.264703	0.480027	0.852955
1	4.9	3.0	1.4	0.2	0	1	-2.080961	-0.674134	0.815495
2	4.7	3.2	1.3	0.2	0	1	-2.364229	-0.341908	0.829315
3	4.6	3.1	1.5	0.2	0	1	-2.299384	-0.597395	0.805014
4	5.0	3.6	1.4	0.2	0	1	-2.389842	0.646835	0.849302

각 점마다 점수 있음

```
irisDF.groupby('cluster')['silhouette_coeff'].mean()
```

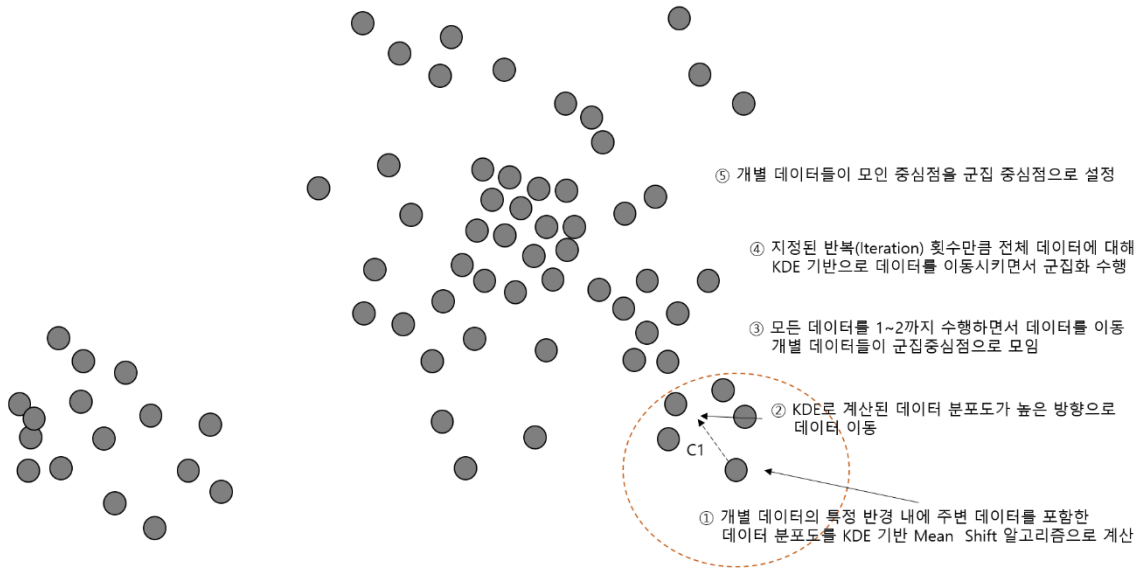
Out :

```
cluster
0    0.417320
1    0.798140
2    0.451105
```

• 평균 이동(Mean Shift)

비모수검정

> K- 평균과 유사하게 중심점을 이동하며 군집화
> 차이점은 K-평균은 평균 거리 중심으로 이동하지만 평균 이동은 분포 밀도가 높은 곳으로 이동한다.
> 주변 데이터와의 거리 값을 KDE(Kernel Density Estimation) 함수 값으로 입력한 뒤 업데이트하며 이동



• 평균 이동(Mean Shift)

> **평균 이동 실습** 집단 갯수 선정할 수 없음

```
from sklearn.cluster import MeanShift

meanshift = MeanShift(bandwidth = 0.8)
cluster_labels = meanshift.fit_predict(irisDF.iloc[:,:4])
print('클러스터 개수 :', len(np.unique(cluster_labels)))
```

bandwidth : 분포도 넓이:
수가 클 수록 많이 포함할 수 있다/ 집단 줄어듦

Out : **클러스터 개수 : 4**

• 평균 이동(Mean Shift)

> 평균 이동 실습

```
from sklearn.cluster import estimate_bandwidth

bandwidth = estimate_bandwidth(irisDF.iloc[:,:4])
print('bandwidth 값:', round(bandwidth, 3))

meanshift = MeanShift(bandwidth = bandwidth)
cluster_labels = meanshift.fit_predict(irisDF.iloc[:,:4])
print('클러스터 개수 :', len(np.unique(cluster_labels)))
```

Out : bandwidth 값: 1.202
클러스터 개수 : 2

• 평균 이동(Mean Shift)

> **평균 이동 실습**

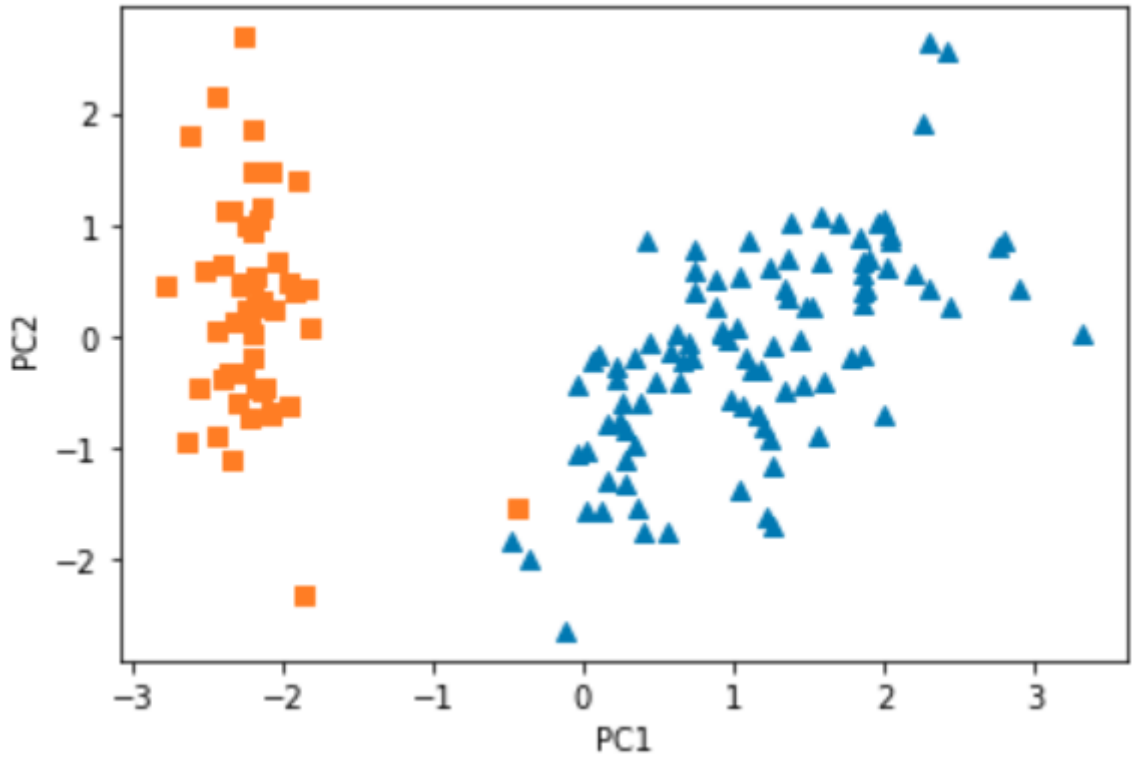
```
markers = ['^', 's']

for i, marker in enumerate(markers):
    x_axis_data = irisDF[irisDF['meanshift'] == i]['PC1']
    y_axis_data = irisDF[irisDF['meanshift'] == i]['PC2']
    plt.scatter(x_axis_data, y_axis_data, marker = marker)

plt.xlabel('PC1')
plt.ylabel('PC2')
plt.show()
```

• 평균 이동(Mean Shift)

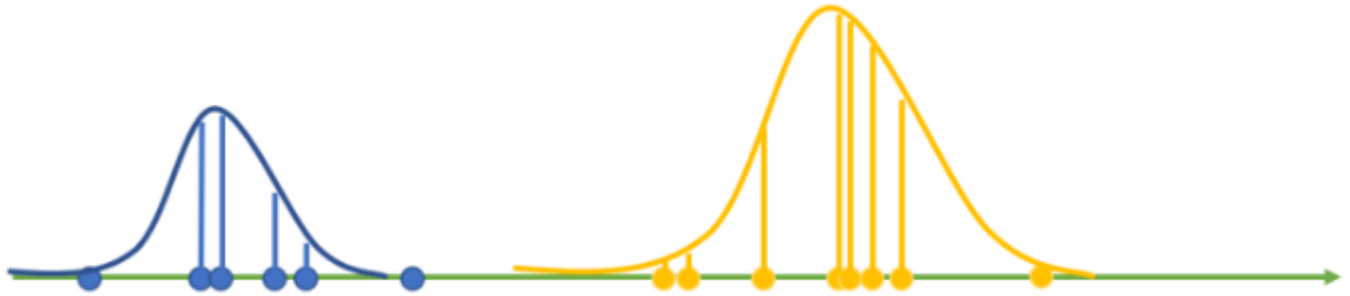
> 평균 이동 실습



- GMM(Gaussian Mixture Model)

모수검정: 정규성 가지는 데이터에 사용

> MeanShift와 유사하게 밀도중심점을 이동하며 군집화
> 차이점은 기본적으로 우리가 가진 데이터들이 모두 정규분포를 따른다고 가정하고 시작
> 최대우도법을 사용하여 데이터의 모수가 어떤 분포를 따를지를 계산하여 군집화



• GMM(Gaussian Mixture Model)

> GMM 실습

```
from sklearn.mixture import GaussianMixture


gmm = GaussianMixture(n_components = 3, random_state = 0)
gmm.fit(irisDF.iloc[:,:4])
gmm_labels = gmm.predict(irisDF.iloc[:,:4])
irisDF['gmm'] = gmm_labels
print(gmm_labels)
```

Out :

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 0 2 0
 0 0 0 2 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

• GMM(Gaussian Mixture Model)

> GMM 실습

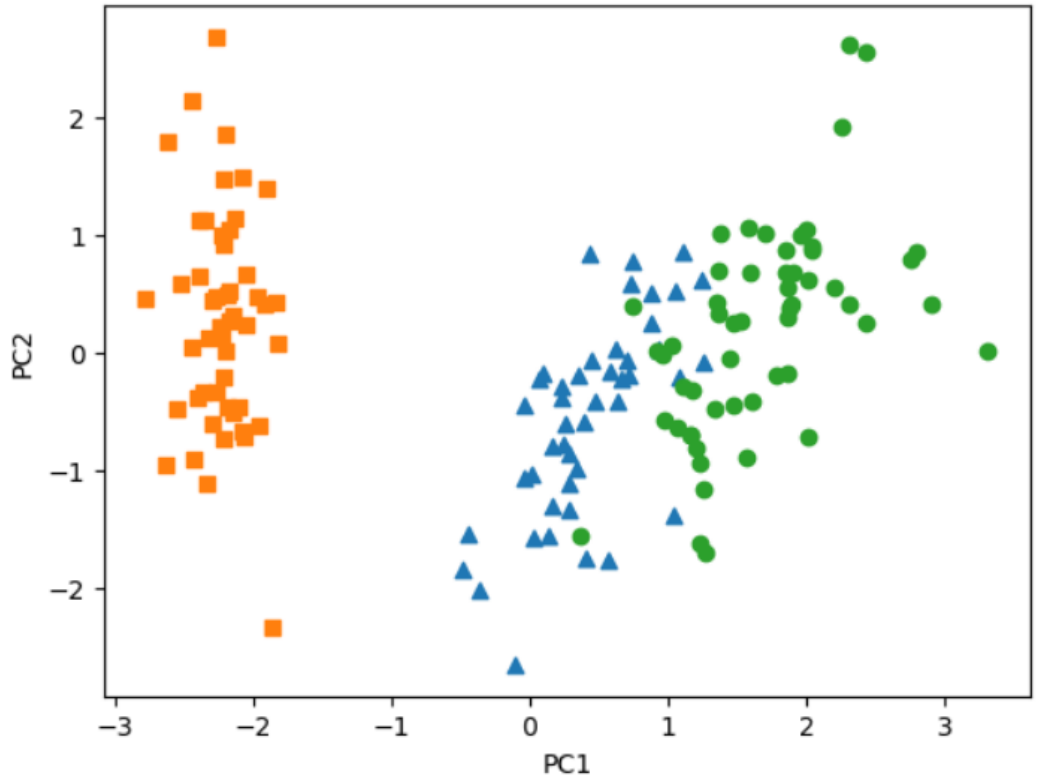
```
markers = ['^', 's', 'o']

for i, marker in enumerate(markers):
    x_axis_data = irisDF[irisDF['gmm'] == i]['PC1']
    y_axis_data = irisDF[irisDF['gmm'] == i]['PC2']
    plt.scatter(x_axis_data, y_axis_data, marker = marker)

plt.xlabel('PC1')
plt.ylabel('PC2')
plt.show()
```

- GMM(Gaussian Mixture Model)

> GMM 실습



gmm.means_ 활용하여 중심점 찍을 수있음

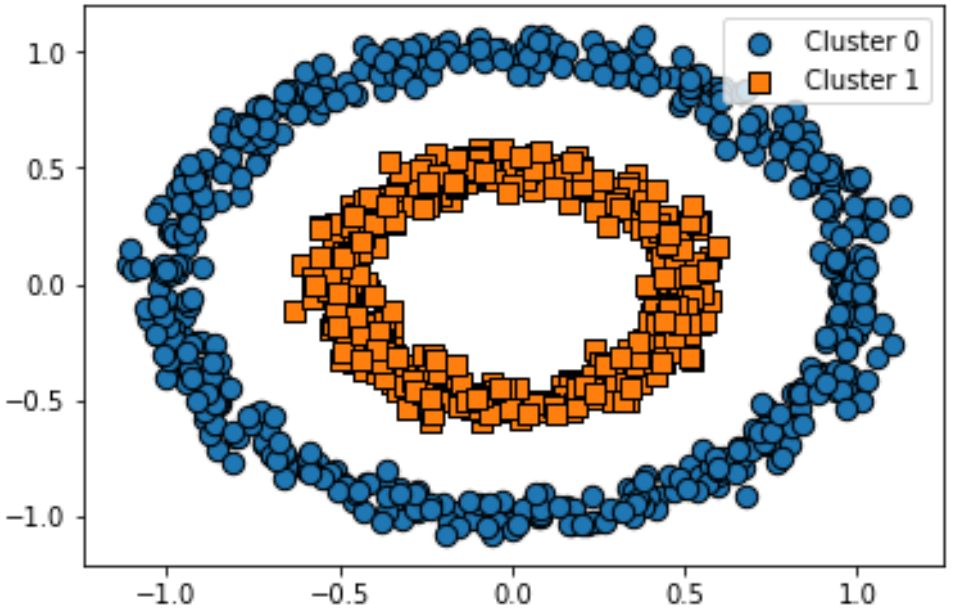
- DBSCAN

  떨어져 있는 데이터에서는 좋음

> 밀도 기반 군집화의 대표적인 알고리즘

> 아래 그림과 같은 형태의 데이터 분포에서는 앞선 군집화가 효과적이지 않다.

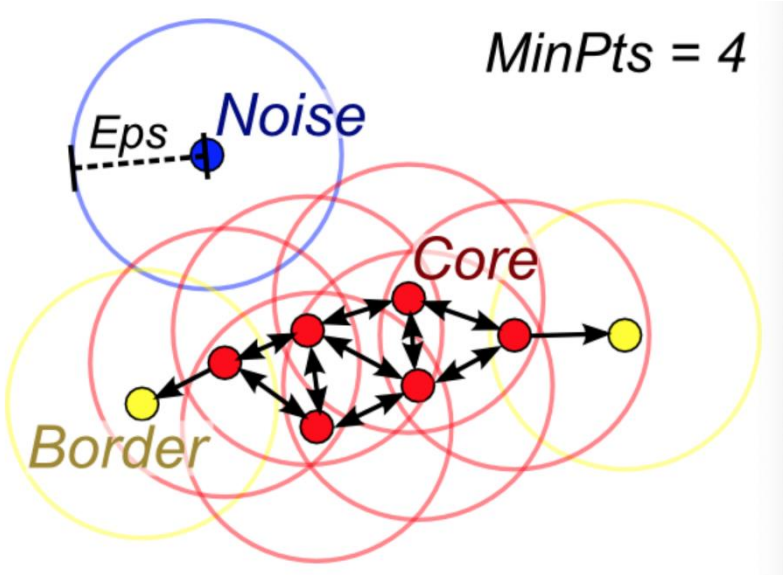
> 복잡한 기하학적 분포도를 가진 데이터에서 효과적이다.



• DBSCAN

> **용어 설명**

- Min points : 군집을 정하기 위한 최소 데이터 개수
- eps : 이웃 점과의 최대 거리(유클리디안)
- core point : 군집의 중심 데이터, 기준점
- border point : 군집에 속하지만 중심점이 되지 않는 데이터
- Noise : 군집에 속하지 못하는 데이터



• DBSCAN

> DBSCAN 실습

```
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

iris = load_iris()
columns = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']

irisDF = pd.DataFrame(data=iris.data, columns = columns)
irisDF['target'] = iris.target
```

• DBSCAN

> DBSCAN 실습

```
from sklearn.cluster import DBSCAN
                                    그룹 갯수 지정 못함: 알아서 계산
dbscan = DBSCAN(eps=0.6, min_samples = 8, metric = 'euclidean')
dbscan_labels = dbscan.fit_predict(iris.data)

irisDF['dbscan_cluster'] = dbscan_labels
irisDF['target'] = iris.target

iris_result = irisDF.groupby(['target'])['dbscan_cluster'].value_counts()
print(iris_result)
```

• DBSCAN

> DBSCAN 실습

```
from sklearn.decomposition import PCA
scaler = StandardScaler()
scaled = scaler.fit_transform(irisDF.iloc[:,:4])
pca = PCA(n_components=2)
pca_transformed = pca.fit_transform(scaled)

irisDF['PC1'] = pca_transformed[:,0]
irisDF['PC2'] = pca_transformed[:,1]
```

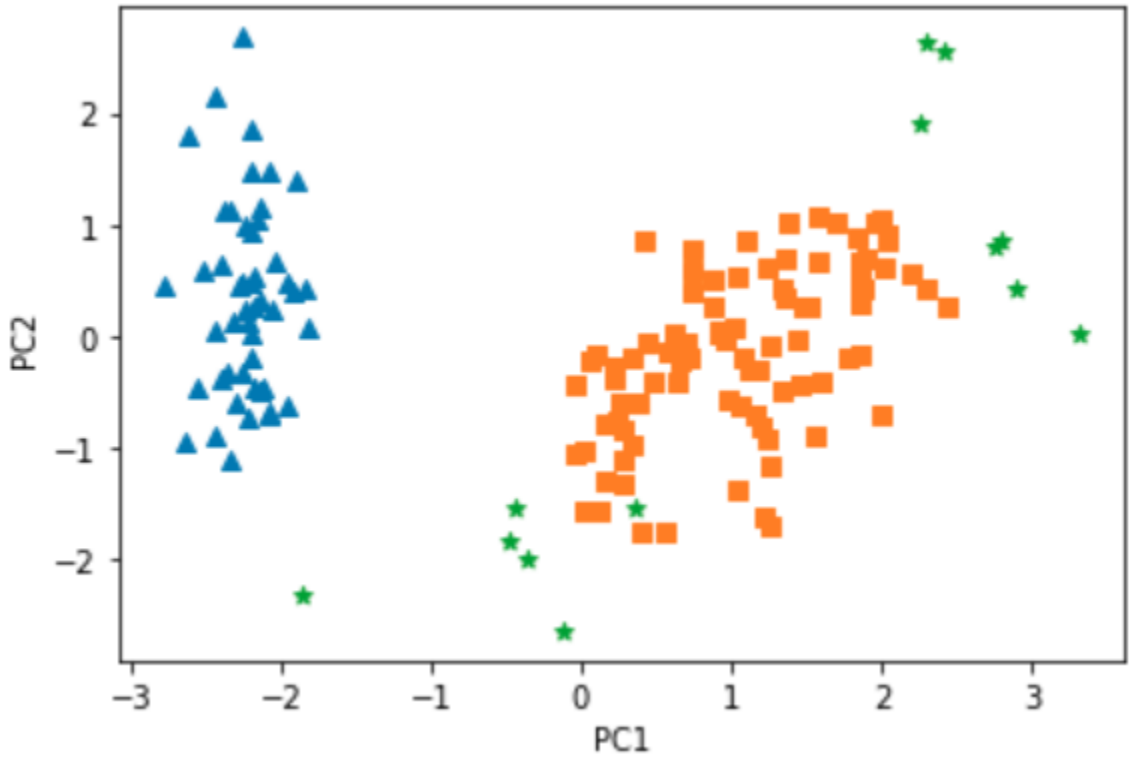
• DBSCAN

> DBSCAN 실습

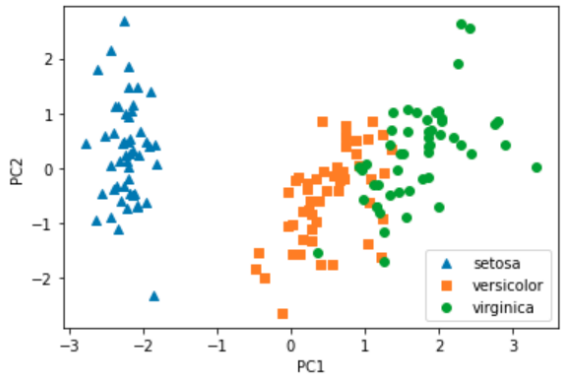
```
markers = ['^', 's']
for i, marker in enumerate(markers):
    x_axis_data = irisDF[irisDF['dbscan_cluster'] == i]['PC1']
    y_axis_data = irisDF[irisDF['dbscan_cluster'] == i]['PC2']
    plt.scatter(x_axis_data, y_axis_data, marker = marker)
# 노이즈 표시
x_axis_data = irisDF[irisDF['dbscan_cluster'] == -1]['PC1']
y_axis_data = irisDF[irisDF['dbscan_cluster'] == -1]['PC2']
plt.scatter(x_axis_data, y_axis_data, marker = '*')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.show()
```

• DBSCAN

> DBSCAN 실습

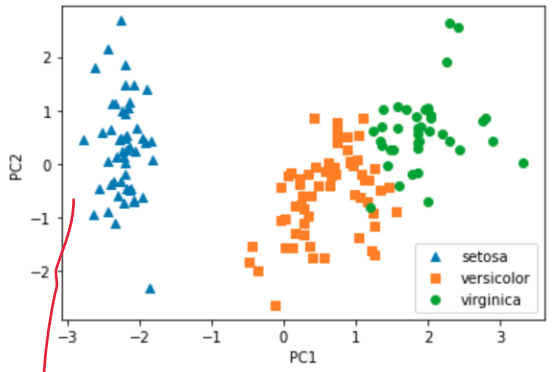


- 군집화 결과 비교

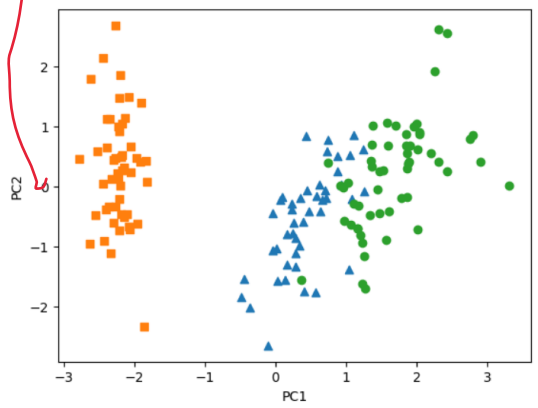


<실제 분류>

군집 알때

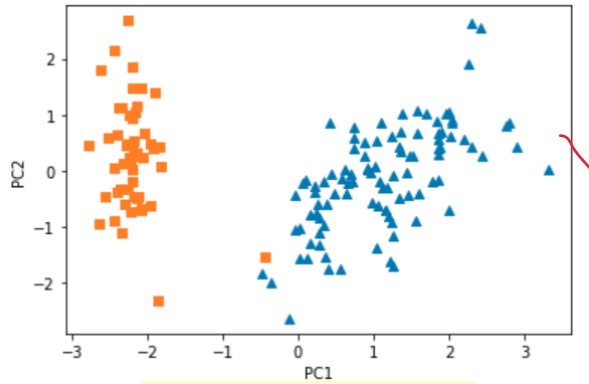


<Kmeans>

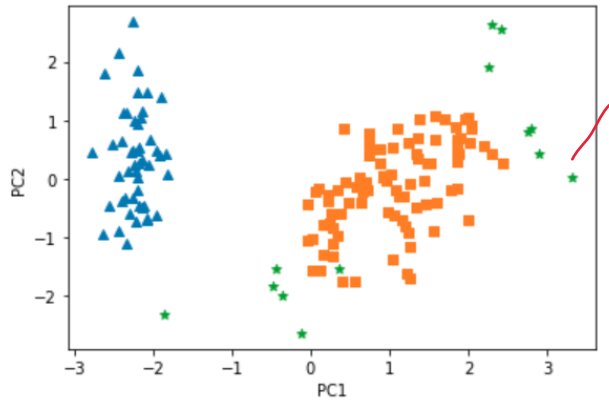


<GMM>

군집 갯수 모를 때



<MeanShift>



<DBSCAN>