UST 과학기술연합대학원대학교
UNIVERSITY OF SCIENCE & TECHNOLOGY

10장

군집화

이홍석 (hsyi@kisti.re.kr)
www.ust.ac.kr

Contents

목차

Contents 1 K-Means 개요

Contents 2 K-Means의 이해

Contents 3 K-Means 상가고객 군집

Contents 4 K-Mmeans 적용 붓꽃

Contents 5 GMM 군집화

Contents 6 DBSCAN 군집화

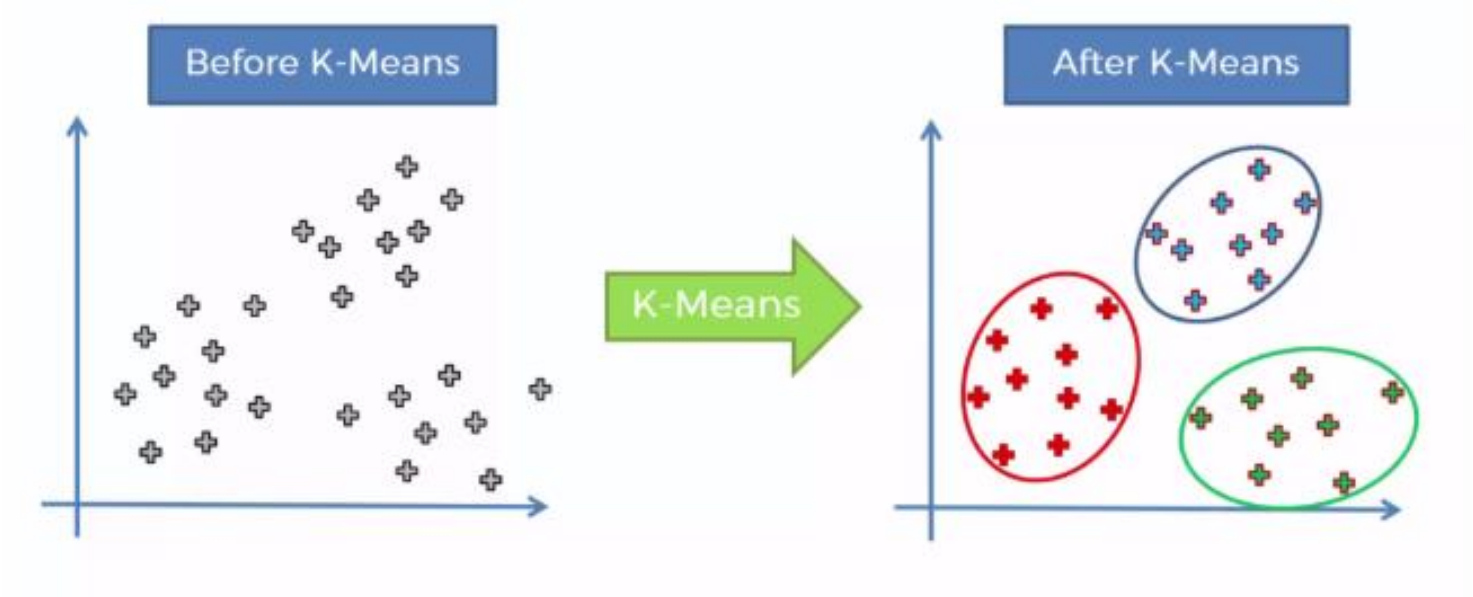
UST 과학기술연합대학원대학교
UNIVERSITY OF SCIENCE & TECHNOLOGY

01.
K-Means 소개

이홍석 (hsyi@kisti.re.kr)

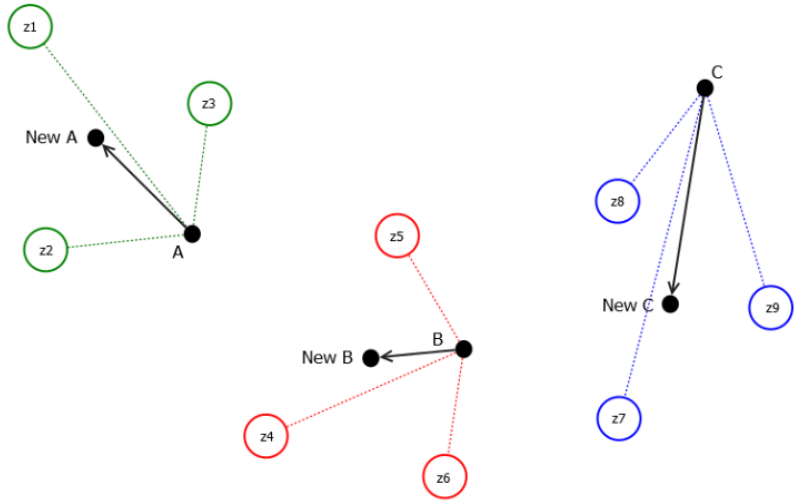
K-Means 개요 (1)

- K-Means는
  - 비지도학습(Unsupervised Learning) 이며 라벨은 없고
  - “k” 값은 클러스터 그룹 수를 의미



K-Means 개요 (2)

- K-평균 알고리즘
  - 중심점(centroid)을 선택하고, 중심에 가장 가까운 포인트를 선택하여 군집화
  - 중심점은 선택된 포인트의 평균지점으로 이동하고, 이동된 중심점에서 다시 가장 가까운 포인트를 선택하며, 다시 중심점을 평균지점으로 이동하고, 이를 반복
  - 모든 데이터 포인트에 더 이상 중심점 이동이 없을 경우에 반복을 멈춤
  - 해당 중심점에 속하는 데이터 포인트를 군집화.



$$J = \sum_{k=1}^{K} \sum_{i \in C_k} d(x_i, \mu_k)$$

군집 개수

중심점 개수

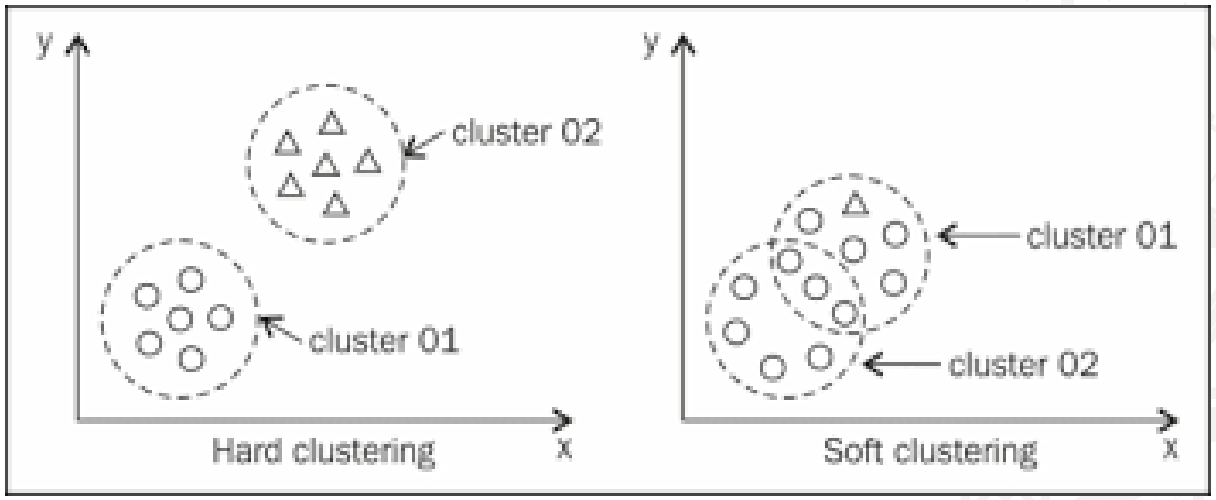
군집에 속하는 데이터 수

$$d(x_i, \mu_k) = ||x_i - \mu_k||^2$$

2개 데이터 사이의 거리

K-Means 개요 (3)

- **군집 분석 방법**
  - 하드 클러스터링 : 하나의 데이터가 정확히 하나의 군집에 할당하는 것
  - 소프트 클러스터링 : 하나의 데이터가 다수의 군집에 할당하는 것



K-Means 개요 (4)

- K-Means 클러스터링 알고리즘
  - 가장 단순하고 빠른 클러스터링 알고리즘의 하나
  - 목적함수 값이 최소화될 때까지 클러스터의 중심점(centroid) 위치와 각 데이터가 소속될 클러스터를 반복해서 찾음
  - 이 값을 inertia라고 함

군집화 할 개수로 군집 중심점의 개수

초기에 군집 중심점의 좌표를 설정할 방식

sklearn.cluster.KMeans

```
class sklearn.cluster.KMeans(n_clusters=8, init='k-means++', n_init=10, max_iter=300,
tol=0.0001, precompute_distances='auto', verbose=0, random_state=None, copy_x=True,
n_jobs=None, algorithm='auto')
```

최대 반복 횟수, 이 횟수 이전에
모든 데이터의 중심점 이동이 없으면 종료

K-Means clustering.

K-Means 개요 (5)

각 군집의 중심점 좌표 (shape는 [군집 개수, 피처 개수)

Attributes:

cluster_centers_ : *ndarray of shape (n_clusters, n_features)*

Coordinates of cluster centers. If the algorithm stops before fully converging (see `tol` and `max_iter`), these will not be consistent with `labels_`.

labels_ : *ndarray of shape (n_samples,)* → 각 데이터 포인트가 속한 군집 중심점 레이블

Labels of each point

inertia_ : *float* → 샘플 거리 제곱의 합

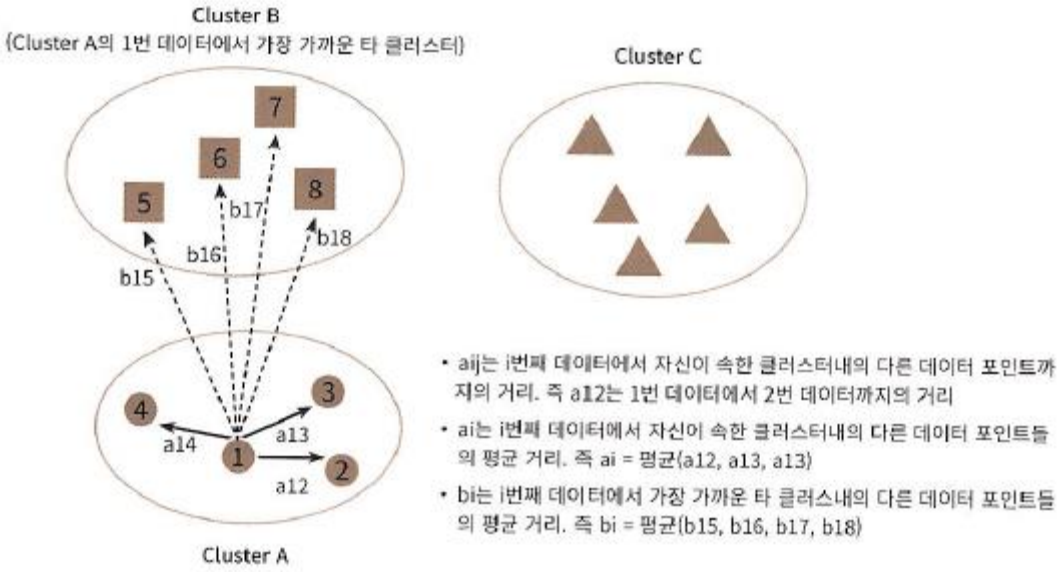
Sum of squared distances of samples to their closest cluster center.

n_iter_ : *int*

Number of iterations run.

K-Means 개요 (6)

실루엣 분석 (Silhouette)



- ai는 i번째 데이터에서 자신이 속한 클러스터내의 다른 데이터 포인트까지의 거리. 즉 a12는 1번 데이터에서 2번 데이터까지의 거리
- ai는 i번째 데이터에서 자신이 속한 클러스터내의 다른 데이터 포인트들의 평균 거리. 즉 ai = 평균(a12, a13, a13)
- bi는 i번째 데이터에서 가장 가까운 타 클러스내의 다른 데이터 포인트들의 평균 거리. 즉 bi = 평균(b15, b16, b17, b18)

K-Means 개요 (7)

실루엣 분석 (Silhouette)

- 각 군집간의 거리가 얼마나 효율적으로 분리 되었는지 나타냄
  - 다른 군집과는 거리가 떨어져 있고, 동일 군집끼리는 잘 뭉쳐 있음
  - 군집화가 잘 되어 있으며, 개별 군집은 비슷한 정도의 여유공간
- 실루엣 분석은 실루엣 계수를 기반
  - 계수는 개별 데이터가 가지는 군집화 지표
  - 해당 데이터가 같은 군집 내의 데이터와 얼마나 가깝게 군집되어 있는가

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

두 군집 간의 거리

두 군집 간의 거리

해당 포인트와 같은 군집 내에 있는 다른 데이터 포인트와의 거리

$$s(i) = \begin{cases} 1 - a(i)/b(i), & \text{if } a(i) < b(i) \\ 0, & \text{if } a(i) = b(i) \\ b(i)/a(i) - 1, & \text{if } a(i) > b(i) \end{cases}$$

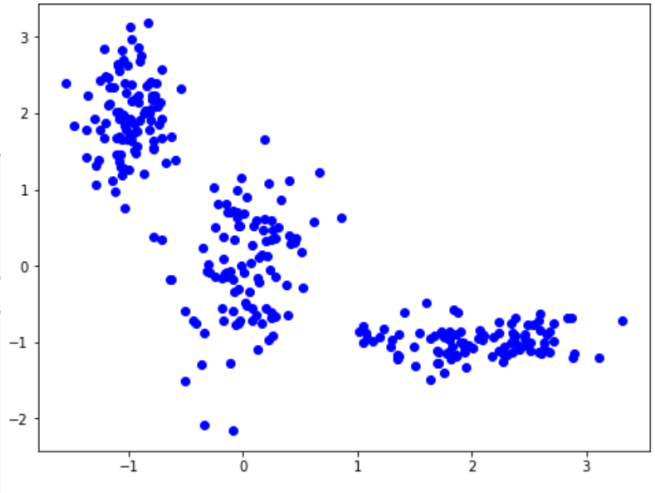
1에 가까울 수록 근처의 군집과 더 멀리 떨러져 있음

UST 과학기술연합대학원대학교
UNIVERSITY OF SCIENCE & TECHNOLOGY

02.
K-Means 예제

이홍석 (hsyi@kisti.re.kr)

K-Means 예제 : 3개 군집 (1)




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

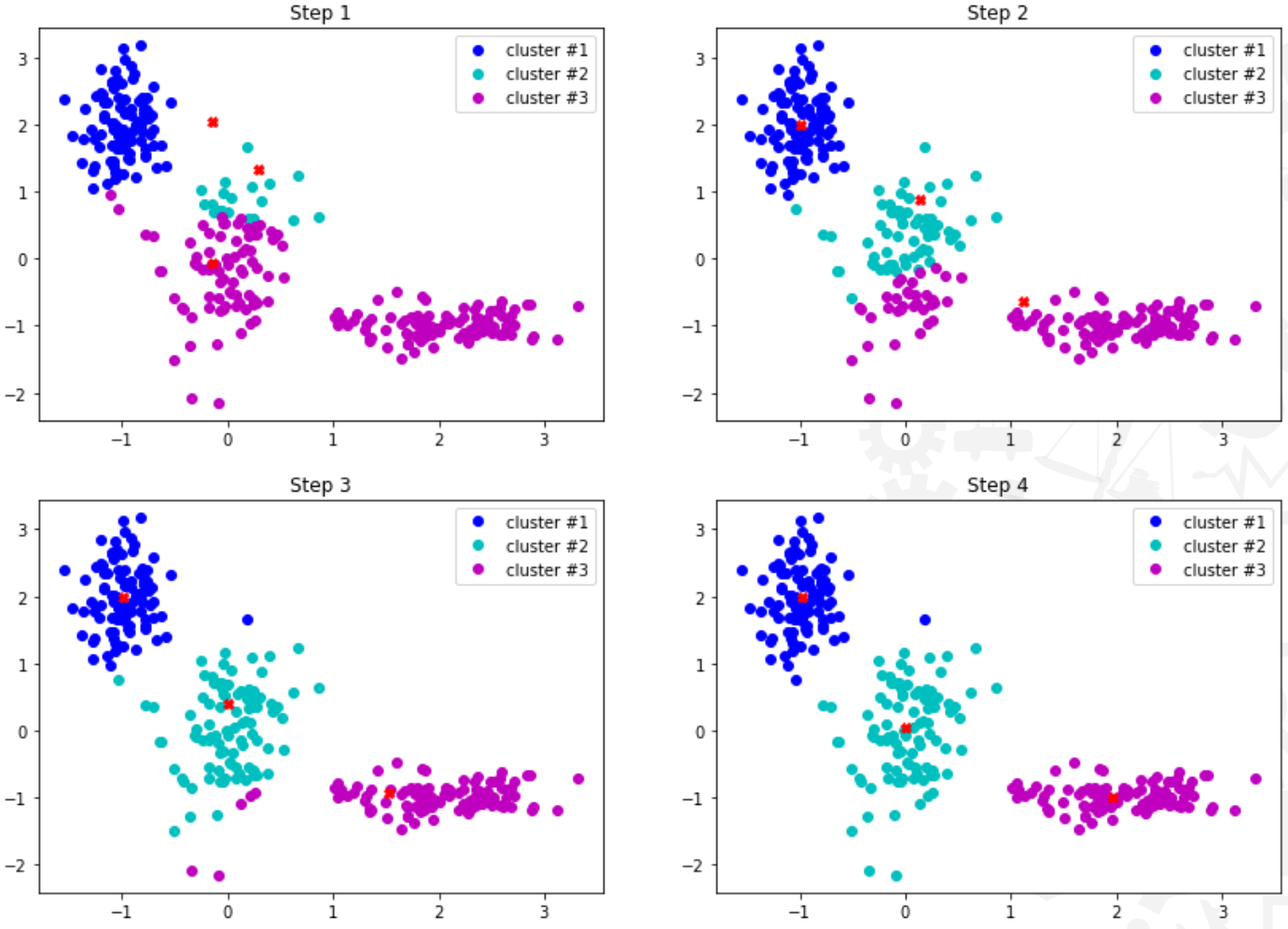
```
# Let's begin by allocation 3 cluster's points
X = np.zeros((300, 2))

np.random.seed(seed=99)
X[:100, 0] = np.random.normal(loc=0.0, scale=.3, size=100)
X[:100, 1] = np.random.normal(loc=0.0, scale=.7, size=100)

X[100:200, 0] = np.random.normal(loc=2.0, scale=.5, size=100)
X[100:200, 1] = np.random.normal(loc=-1.0, scale=.2, size=100)

X[200:300, 0] = np.random.normal(loc=-1.0, scale=.2, size=100)
X[200:300, 1] = np.random.normal(loc=2.0, scale=.5, size=100)

plt.figure(figsize=(8, 6))
plt.plot(X[:, 0], X[:, 1], 'bo');
```

K-Means 예제 : 3개 군집 (2)

```
for i in range(3):
    # 클러스터 중심에서 각각의 점들의 거리를 계산
    distances = cdist(X, centroids)
    # 가장 가까운 중심점 확인
    labels = distances.argmin(axis=1)

    # 거리에 따라서 라베링하기
    centroids = centroids.copy()
    centroids[0, :] = np.mean(X[labels == 0, :], axis=0)
    centroids[1, :] = np.mean(X[labels == 1, :], axis=0)
    centroids[2, :] = np.mean(X[labels == 2, :], axis=0)

    cent_history.append(centroids)
```

```
for i in range(4):
    distances = cdist(X, cent_history[i])
    labels = distances.argmin(axis=1)

    plt.subplot(2, 2, i + 1)
    plt.plot(X[labels == 0, 0], X[labels == 0, 1], 'bo', label='cluster #1')
    plt.plot(X[labels == 1, 0], X[labels == 1, 1], 'co', label='cluster #2')
    plt.plot(X[labels == 2, 0], X[labels == 2, 1], 'mo', label='cluster #3')
    plt.plot(cent_history[i][:, 0], cent_history[i][:, 1], 'rX')
    plt.legend(loc=0)
    plt.title('Step {:}'.format(i + 1));
```

K-Means 예제 : 3개 군집 (3)



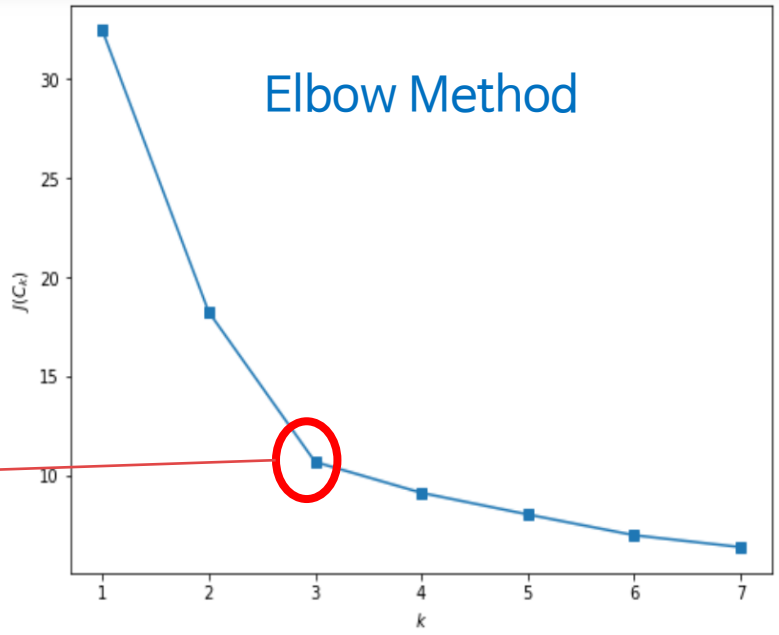
K-Means 예제 : 3개 군집 (4)

```
from sklearn.cluster import KMeans
```

```
error = []
for k in range(1, 8):
    kmeans = KMeans(n_clusters=k, random_state=1).fit(X)
    error.append(np.sqrt(kmeans.inertia_))
    # 중간과정을 확인하고 싶으로 아래의 주석
    #print(k,np.square(kmeans.inertia_))
```

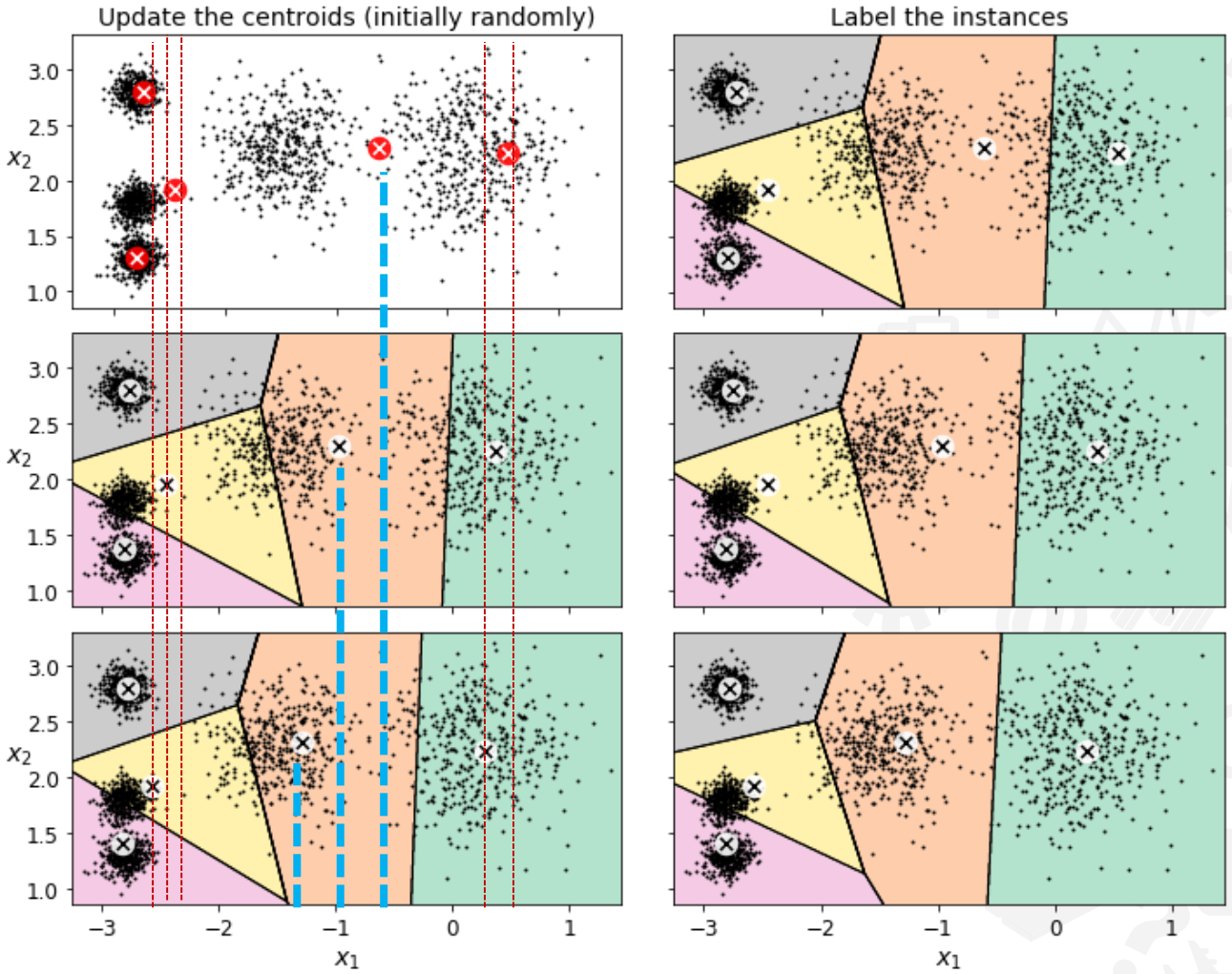
```
plt.figure(figsize=(8, 6))
plt.plot(range(1, 8), error, marker='s');
plt.xlabel('$k$')
plt.ylabel('$J(C_k)$');
```

k=3 이후에는 오브젝트 함수 J의 급격한 변화가 없음.
그래서 3인 최적의 k 값임.



Elbow 방법은 최적의 k 값을 구함

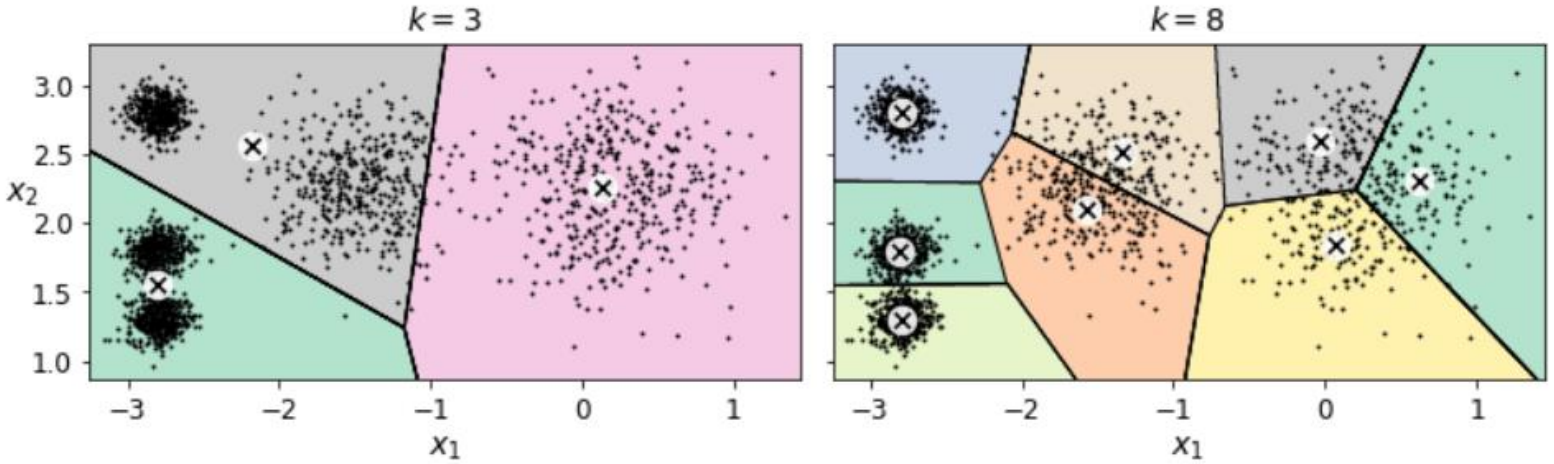
K-Means 예제 : 5개 군집 (5)



K-Means 예제 : 5개 군집 (6)

- K-Means의 최상의 모델 평가를 위해서는
  - 비지도학습, 레이블이 없는 관계로 어렵다
  - 하지만 중심점에 대하 거리를 알고 있어서 Inertia를 사용
- Inertia Inertia ~ 최적의 모델을 주는 k는
  - 각각의 데이터와 가장 가까운 중심점 제곱의 거리

Inertia를 최소화 하는 k를 찾는 것은 쉽지가 않다. k가 많을 수록 inertia는 작다.



K-Means 예제 : 5개 군집 (7)

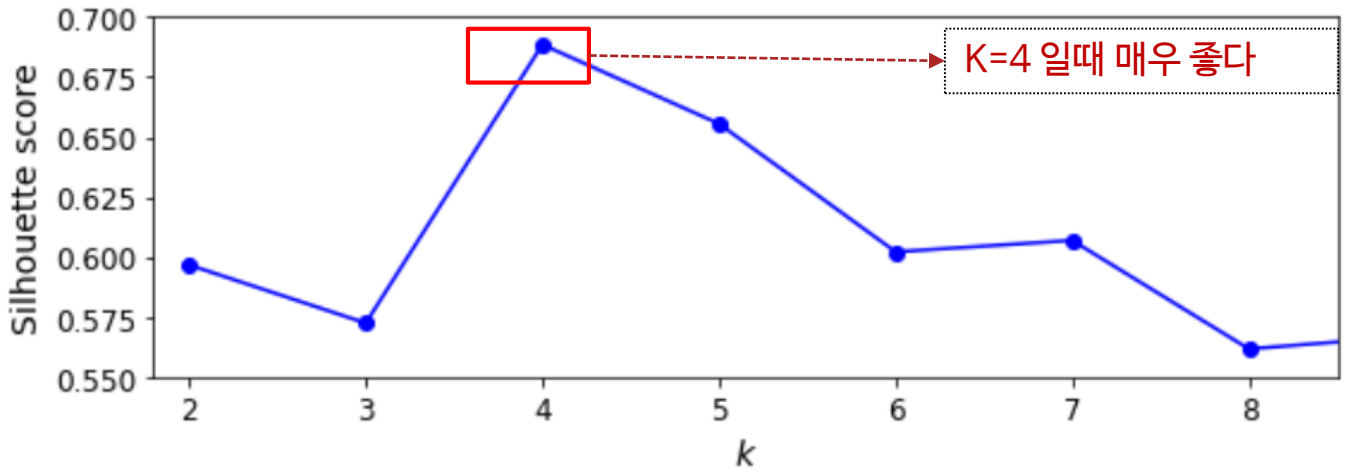
```
from sklearn.metrics import silhouette_score
```

```
silhouette_score(X, kmeans.labels_)
```

```
0.655517642572828
```

실루엣 계수 스코어 0.656 (k=5)

```
silhouette_scores = [silhouette_score(X, model.labels_)
                     for model in kmeans_per_k[1:]]
```

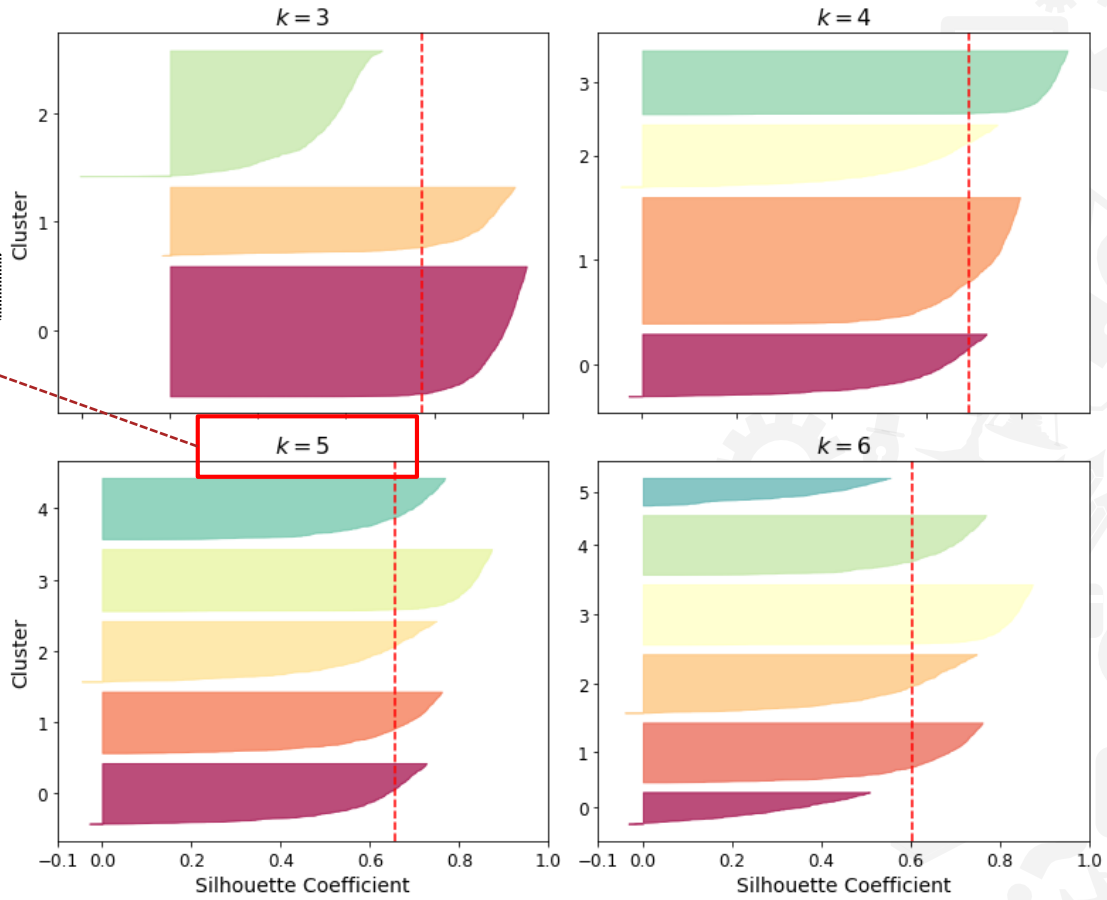


K-Means 예제 : 5개 군집 (7)

점선은 실루엣 스코어로 점선보다 왼쪽에 있으면, 너무 가까워서 BAD!

k=5, 클러스터는 대부분 비슷한 크기이면 모두 점선 오른쪽, 최적의 k로 봄

최적의 분류 k=5



UST 과학기술연합대학원대학교
UNIVERSITY OF SCIENCE & TECHNOLOGY

03. K-Means 적용
상가 고객 데이터

이홍석 (hsyi@kisti.re.kr)

Kmeans-예제: 상가 고객 군집화(1)

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# Importing the dataset
dataset = pd.read_csv('./input/Mall_Customers.csv',index_col='CustomerID')

dataset.head()
```

CustomerID	Genre	Age	Annual_Income_(k$)	Spending_Score
1	Male	19	15	39
2	Male	21	15	81
3	Female	20	16	6
4	Female	23	16	77
5	Female	31	17	40

Kmeans-예제: 상가 고객 군집화(2)

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 200 entries, 1 to 200
Data columns (total 4 columns):
Genre                   200 non-null object
Age                     200 non-null int64
Annual_Income_(k$)      200 non-null int64
Spending_Score          200 non-null int64
dtypes: int64(3), object(1)
memory usage: 7.8+ KB
```

```
dataset.isnull().sum()
```

```
Genre                   0
Age                     0
Annual_Income_(k$)      0
Spending_Score          0
dtype: int64
```

```
dataset.describe()
```

	Age	Annual_Income_(k$)	Spending_Score
count	200.000000	200.000000	200.000000
mean	38.850000	60.560000	50.200000
std	13.969007	26.264721	25.823522
min	18.000000	15.000000	1.000000
25%	28.750000	41.500000	34.750000
50%	36.000000	61.500000	50.000000
75%	49.000000	78.000000	73.000000
max	70.000000	137.000000	99.000000

Kmeans-예제: 상가 고객 군집화(3)

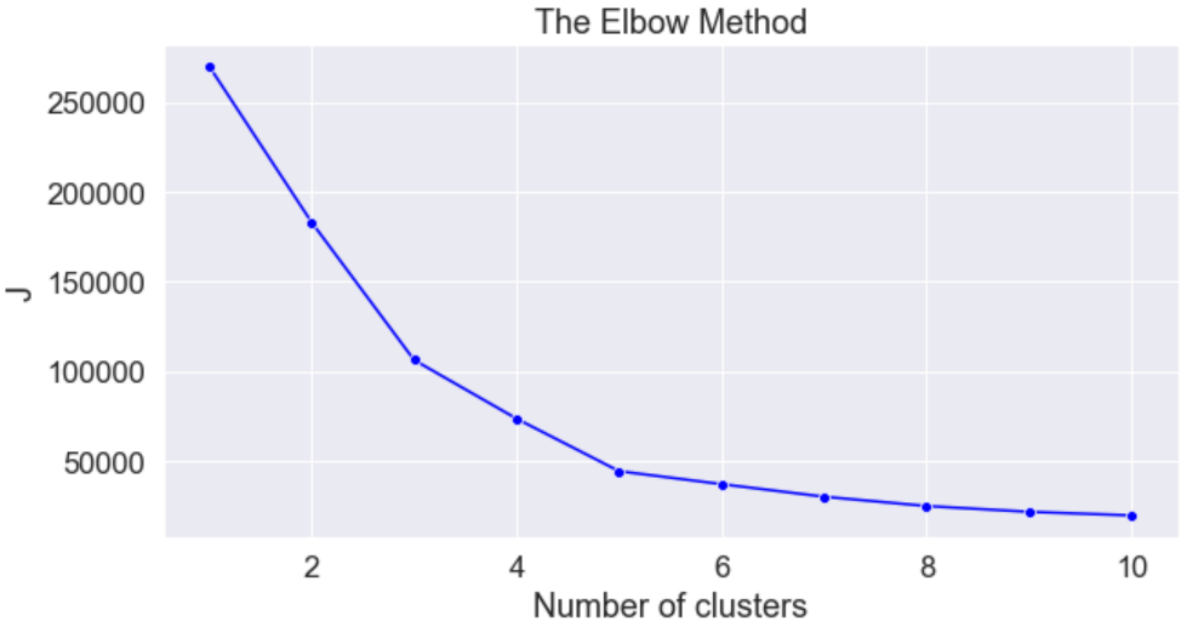
```
dataset.drop_duplicates(inplace=True)
```

```
# using only Spending_Score and income variable for easy visualisation
X = dataset.iloc[:, [2, 3]].values
```

```
# Using the elbow method to find the optimal number of clusters
from sklearn.cluster import KMeans
obj = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 9)
    kmeans.fit(X)
    # inertia method returns wcss for that model
    obj.append(kmeans.inertia_)
```

Kmeans-예제: 상가 고객 군집화(4)

```
plt.figure(figsize=(10,5))
sns.lineplot(range(1, 11), obj,marker='o',color='b')
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('J')
plt.show()
```



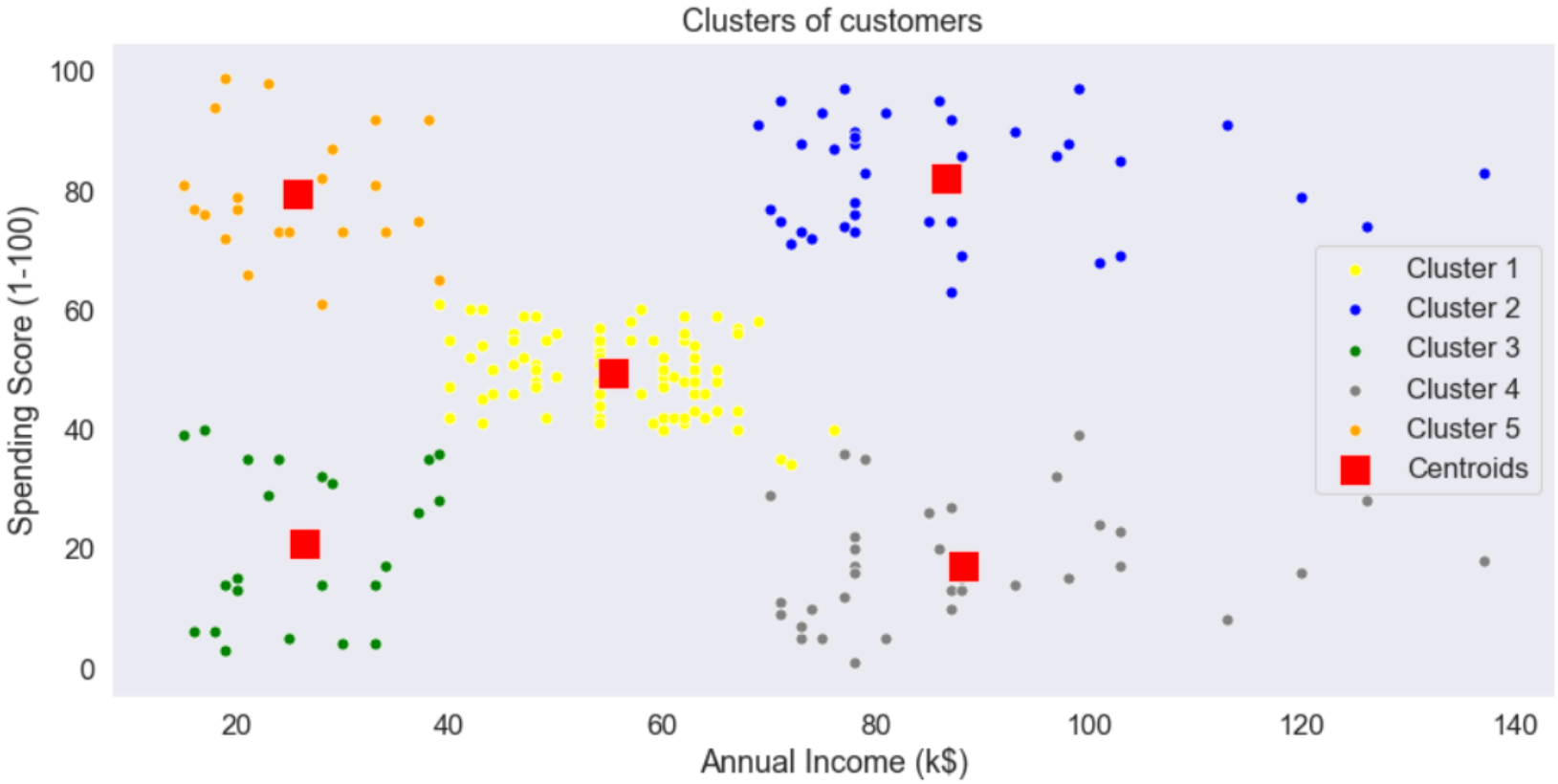
Kmeans-예제: 상가 고객 군집화(5)

```
# Fitting K-Means to the dataset
kmeans = KMeans(n_clusters = 5, init = 'k-means++', random_state = 9)
y_kmeans = kmeans.fit_predict(X)
```

7. Visualisation

```
# Visualising the clusters
plt.figure(figsize=(15,7))
sns.scatterplot(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], color = 'yellow', label = 'Cluster 1',s=50)
sns.scatterplot(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], color = 'blue', label = 'Cluster 2',s=50)
sns.scatterplot(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], color = 'green', label = 'Cluster 3',s=50)
sns.scatterplot(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], color = 'grey', label = 'Cluster 4',s=50)
sns.scatterplot(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], color = 'orange', label = 'Cluster 5',s=50)
sns.scatterplot(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], color = 'red',
                label = 'Centroids',s=300,marker=',')
plt.grid(False)
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```

Kmeans-예제: 상가 고객 군집화(6)



UST 과학기술연합대학원대학교
UNIVERSITY OF SCIENCE & TECHNOLOGY

04. K-Means 적용
붓꽃

이홍석 (hsyi@kisti.re.kr)

K-Means for Iris (1)

K-Means를 이용한 붓꽃(Iris) 데이터 셋 Clustering

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

```
iris = load_iris()

df_iris = pd.DataFrame(data=iris.data, columns=['sepal_length',
                                                'sepal_width','petal_length','petal_width'])
df_iris.head(3)
```

꽃받침 길이

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

꽃잎 넓이

K-Means for Iris (2)

```
kmeans = KMeans(n_clusters=3, init='k-means++', max_iter=300,random_state=0).fit(df_iris)
```

붓꽃은 3종으로 n_clusters=3으로 설정이 올바르고, 군집화 초기 중심점 설정 방식은 디폴트 값인 'k-means++', 최대반복횟수=300 객체 만든후 fit()함

```
print(kmeans.labels_)
```

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

Kmeans.fit()을 수행한 결과 labels_ 속성값을 df_iris 'cluster' 컬럼으로 추가함

```
df_iris['cluster']=kmeans.labels_
```

	sepal_length	sepal_width	petal_length	petal_width	cluster
0	5.1	3.5	1.4	0.2	1
1	4.9	3.0	1.4	0.2	1
2	4.7	3.2	1.3	0.2	1
3	4.6	3.1	1.5	0.2	1
4	5.0	3.6	1.4	0.2	1

Kmeans 클러스터링 결과로 Labels_는 0, 1, 2로 군집화

K-Means for Iris (3)

실제 붓꽃 품종 분류 값 타켓 라벨과 군집화 분류 값인 'cluster' 컬럼(라벨) 적용

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

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

타겟 1일때 kmeans 결과는 2개만 0번 나머지는 48개는 2번

Target 2일 경우, 0번 군집은 36개 2번 군집은 14개

K-Means for Iris (4)

붓꽃의 시각화는 4개의 속성을 2개의 차원으로 줄이는 PCA 기법 적용

```
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
pca_transformed = pca.fit_transform(iris.data)

df_iris['pca_1'] = pca_transformed[:,0]
df_iris['pca_2'] = pca_transformed[:,1]
df_iris.head(3)
```

2개의 차원으로 축소

	sepal_length	sepal_width	petal_length	petal_width	cluster	target	pca_1	pca_2
0	5.1	3.5	1.4	0.2	1	0	-2.684126	0.319397
1	4.9	3.0	1.4	0.2	1	0	-2.714142	-0.177001
2	4.7	3.2	1.3	0.2	1	0	-2.888991	-0.144949

Pca_1 주성분 일까 확인필요?

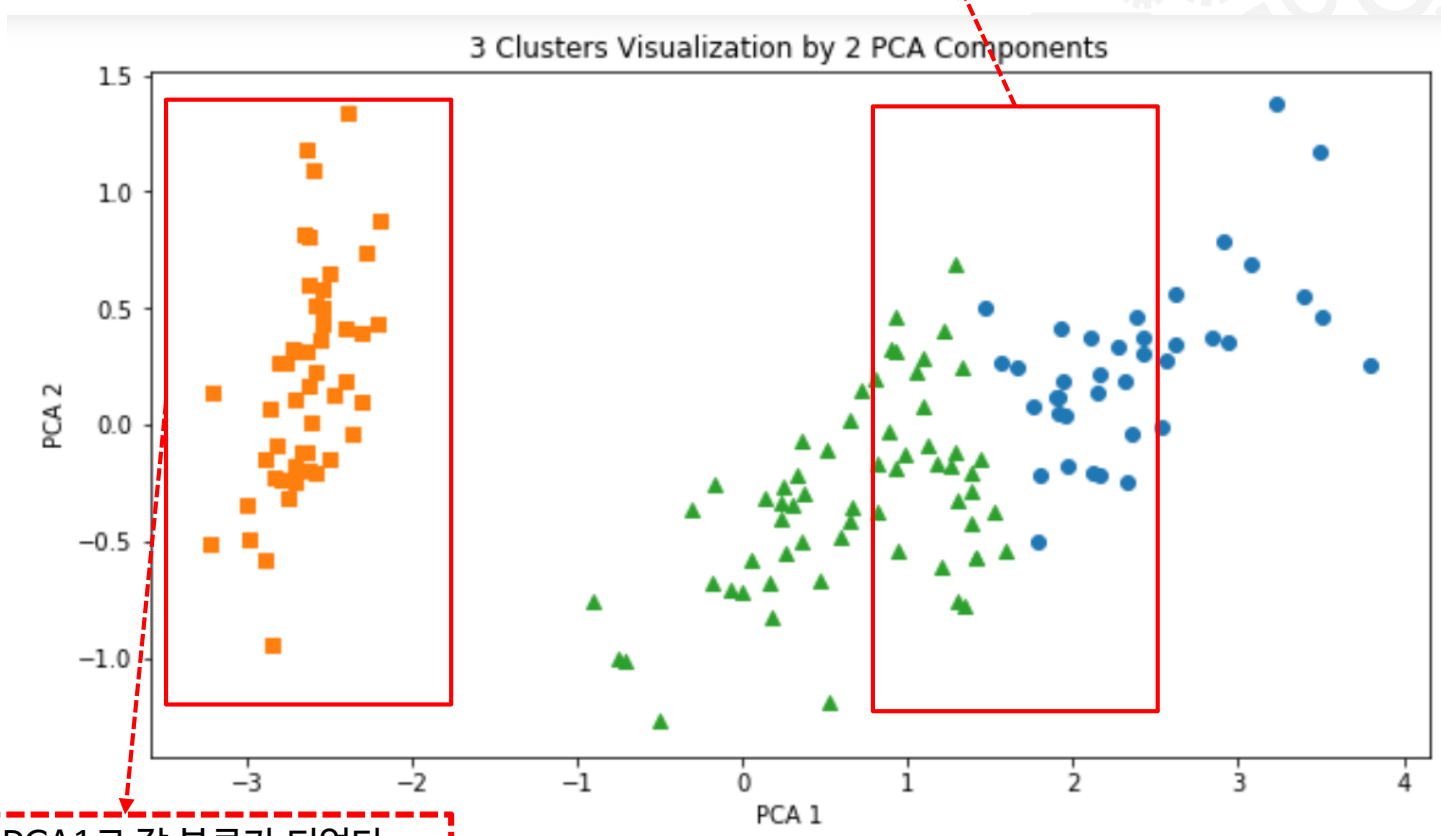
K-Means for Iris (5)

```
# cluster 값이 0, 1, 2 인 경우마다 별도의 Index로 추출
marker0_ind = df_iris[df_iris['cluster']==0].index
marker1_ind = df_iris[df_iris['cluster']==1].index
marker2_ind = df_iris[df_iris['cluster']==2].index

plt.figure(figsize=(10, 5.5))

# cluster값 0, 1, 2에 해당하는 Index로 각 cluster 레벨의 pca_x, pca_y 값 추출. o, s, ^ 로 marker 표시
plt.scatter(x=df_iris.loc[marker0_ind,'pca_1'], y=df_iris.loc[marker0_ind,'pca_2'], marker='o')
plt.scatter(x=df_iris.loc[marker1_ind,'pca_1'], y=df_iris.loc[marker1_ind,'pca_2'], marker='s')
plt.scatter(x=df_iris.loc[marker2_ind,'pca_1'], y=df_iris.loc[marker2_ind,'pca_2'], marker='^')

plt.xlabel('PCA 1')
plt.ylabel('PCA 2')
plt.title('3 Clusters Visualization by 2 PCA Components')
plt.show()
```

'o' 는 동그라미
's' 는 네모
'^' 는 세모 마크

K-Means for Iris (6)

클러스터 0과 1의 경우 속성의 위치가
명확히 분리되기 어려운 부분이 있다.



네모 모양은 PCA1로 잘 분류가 되었다.

K-Means for Iris (7)

실루엣 계수 구하기

```
from sklearn.metrics import silhouette_samples, silhouette_score
```

```
score_samples = silhouette_samples(iris.data,df_iris['cluster'])
```

```
print('silhouette_samples() return shape',score_samples)
```

```
silhouette_samples() return shape [0.85295506 0.81549476 0.8293151  0.
 0.82165093 0.85390505 0.75215011 0.825294   0.80310303 0.83591262
```

K-Means for Iris (8)

실루엣 계수 구하기

```
from sklearn.metrics import silhouette_samples, silhouette_score
```

```
score_samples = silhouette_samples(iris.data,df_iris['cluster'])
```

```
print('silhouette_samples() return shape',score_samples.shape)
```

```
silhouette_samples() return shape (150,)
```

```
df_iris['silhouette_coeff']= score_samples
```

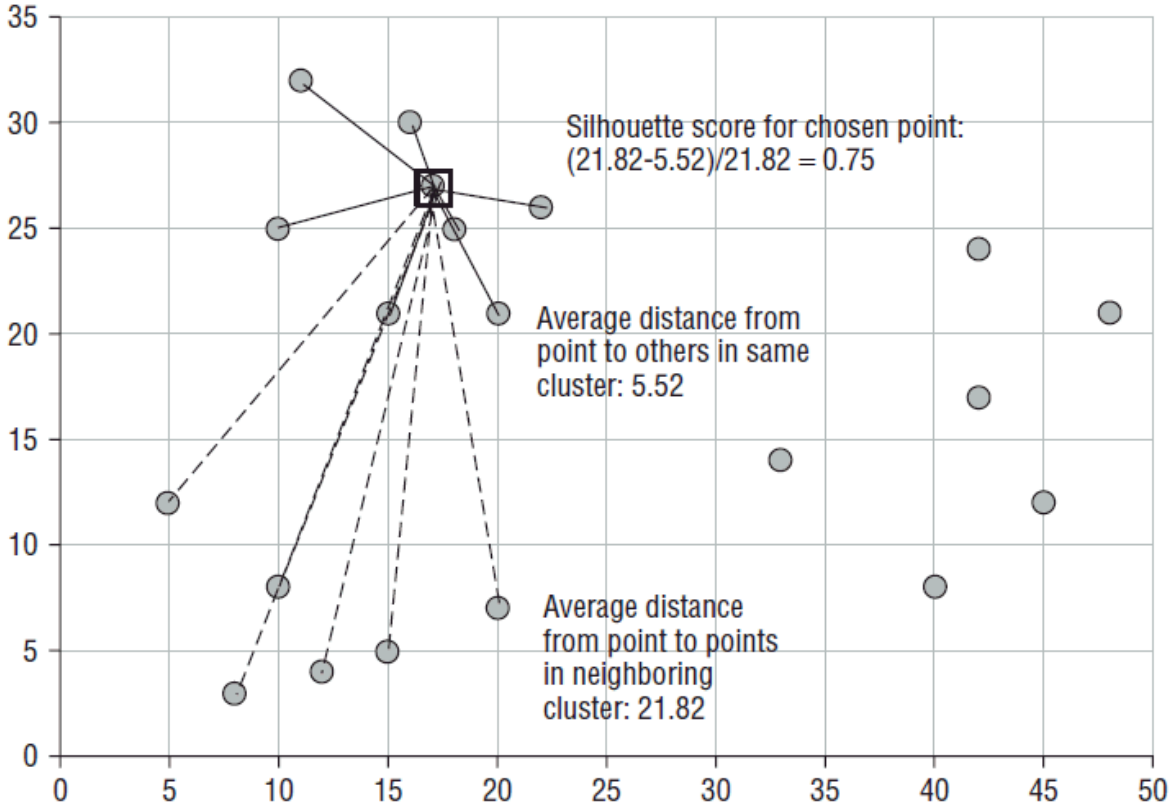
```
average_score=silhouette_score(iris.data, df_iris['cluster'])
print('iris silhouette analysis score:{0:.3f} '.format(average_score))
```

```
iris silhouette analysis score:0.553
```

평균 값이 0.553, 오른쪽 실루엣 계수는 0.852.
다른 군집에서 평균 이하 계수가 있어야 한다.

pca_1	pca_2	silhouette_coeff
-2.684126	0.319397	0.852955

K-Means for Iris (10)



K-Means for Iris (11)

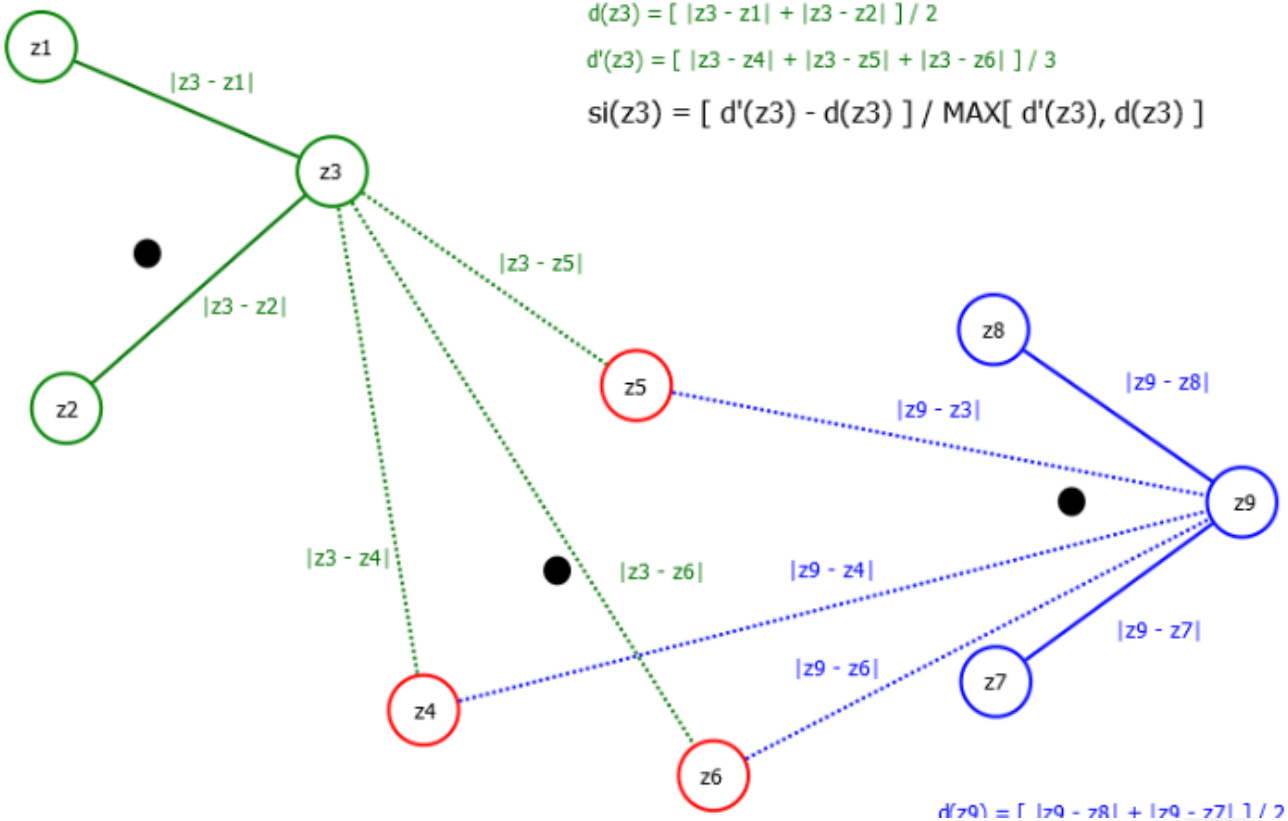
Silhouette Index

SI = { for all zi in Z }, SUM(si(zi)) / |Z|

d(z3) = [|z3 - z1| + |z3 - z2|] / 2

d'(z3) = [|z3 - z4| + |z3 - z5| + |z3 - z6|] / 3

si(z3) = [d'(z3) - d(z3)] / MAX[d'(z3), d(z3)]



K-Means for Iris (9)

```
df_iris.head()
```

	sepal_length	sepal_width	petal_length	petal_width	cluster	target	pca_1	pca_2	silhouette_coeff
0	5.1	3.5	1.4	0.2	1	0	-2.684126	0.319397	0.852955
1	4.9	3.0	1.4	0.2	1	0	-2.714142	-0.177001	0.815495
2	4.7	3.2	1.3	0.2	1	0	-2.888991	-0.144949	0.829315
3	4.6	3.1	1.5	0.2	1	0	-2.745343	-0.318299	0.805014
4	5.0	3.6	1.4	0.2	1	0	-2.728717	0.326755	0.849302

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

```
cluster
0    0.451105
1    0.798140
2    0.417320
Name: silhouette_coeff, dtype:
```

클러스터 0은 0.45,
클러스터 1은 0.79
클러스터 2는 0.41 계수로 평균은 0.553 을 얻음

실루엣 계수는 0~1 사이의 값을 갖지만, 1에 가까울 수록 좋다.
실루엣 계수는 개별 데이터가 가지는 군집화 지표로,
해당 데이터가 같은 군집 내에 데이터와 얼마나 가깝게 군집화 되었는가를 보며, 다른 군집에 있는 데이터와는 얼마나 멀리 분리되어 있는지 나타냄

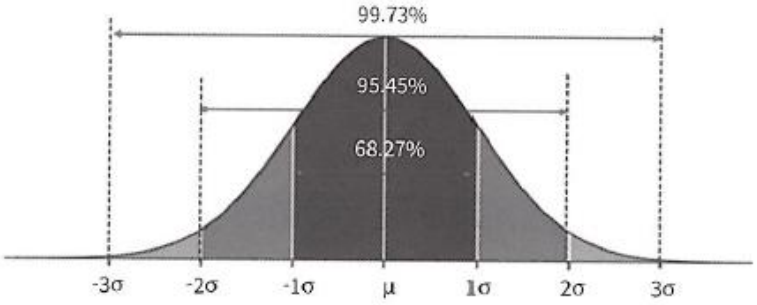
UST 과학기술연합대학원대학교
UNIVERSITY OF SCIENCE & TECHNOLOGY

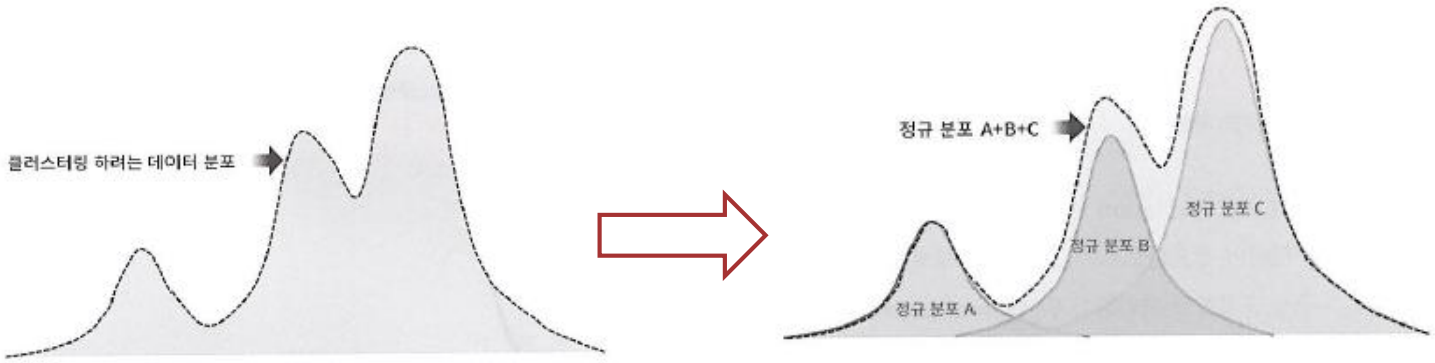
05.
GMM 소개

이홍석 (hsyi@kisti.re.kr)

GMM 소개(1)

- GMM(Gaussian Mixture Model)
  - 군집화를 적용하고자 하는 데이터가 여러 개의 가우시안 분포를 섞어서 생성된 모델로 가정하고 수행하는 방식. 가우시안 모델은 정규분포로 알려짐





GMM을 이용한 붓꽃 군집화(1)

GMM 을 이용한 붓꽃 데이터 셋 클러스터링

```
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
%matplotlib inline

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

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

```
from sklearn.mixture import GaussianMixture

gmm = GaussianMixture(n_components=3, random_state=0).fit(iris.data)

gmm_cluster_labels = gmm.predict(iris.data)
print(gmm_cluster_labels)
```

n_components는 가우시안 mixture 총 수

모델 핏트(훈련시행)

gmm 훈련 결과를 예측함

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

GMM을 이용한 붓꽃 군집화(2)

```
risDF['gmm_cluster'] = gmm_cluster_labels
risDF['target'] = iris.target


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

군집화 결과를 df_iris의 'gmm_cluster' 컬럼으로 저장, 타겟도 마찬가지

target 값에 따라 gmm_cluster 값이 어떻게 매핑 되는지 확인

```
target  gmm_cluster
0       0              50
1       1              45
        2               5
2       2              50
Name: gmm_cluster, dtype: int64
```

target 1은 gmm_cluster 1과 2로 각각 45개와 5개로 매핑됨

```
kmeans = KMeans(n_clusters=3, init='k-means++', max_iter=300,random_state=0).fit(iris.data)

kmeans_cluster_labels = kmeans.predict(iris.data)

irisDF['kmeans_cluster'] = kmeans_cluster_labels

iris_result = irisDF.groupby(['target'])['kmeans_cluster'].value_counts()

print(iris_result)
```

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

K-Means 군집화 평균.
GMM과 비교하여 어느것이 성능이 더 우수한가?

K-Means은 원형으로 분포된 데이터에 우수함.

GMM을 이용한 붓꽃 군집화(3)

```
def visualize_cluster_plot(clusterobj, dataframe, label_name, iscenter=True):
    if iscenter :
        centers = clusterobj.cluster_centers_

    unique_labels = np.unique(dataframe[label_name].values)
    markers=['o', 's', '^', 'x', '*']
    isNoise=False

    for label in unique_labels:
        label_cluster = dataframe[dataframe[label_name]==label]
        if label == -1:
            cluster_legend = 'Noise'
            isNoise=True
        else :
            cluster_legend = 'Cluster '+str(label)

        plt.scatter(x=label_cluster['ftr1'], y=label_cluster['ftr2'], s=70,₩
                    edgecolor='k', marker=markers[label], label=cluster_legend)

        if iscenter:
            center_x_y = centers[label]
            plt.scatter(x=center_x_y[0], y=center_x_y[1], s=250, color='white',
                        alpha=0.9, edgecolor='k', marker=markers[label])
            plt.scatter(x=center_x_y[0], y=center_x_y[1], s=70, color='k',₩
                        edgecolor='k', marker='$%d$' % label)
    if isNoise:
        legend_loc='upper center'
    else: legend_loc='upper right'

    plt.legend(loc=legend_loc)
    plt.show()
```

군집 중심좌표 제공이면 True

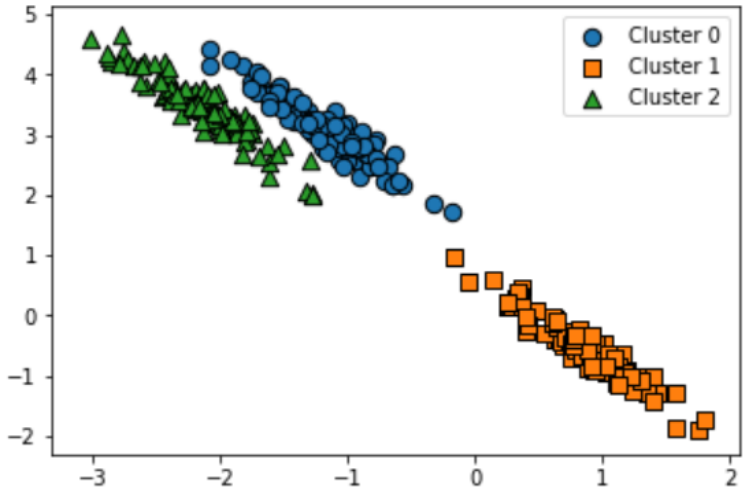
군집 시각화를 위한 라이브러리 만듬

사이킷런의 군집 수행 객체. K-Means나 GMM의 fit()와 predict()로 군집화를 완료한 객체.

GMM을 이용한 붓꽃 군집화(4)

```
from sklearn.datasets import make_blobs


X, y = make_blobs(n_samples=300, n_features=2, centers=3, cluster_std=0.5, random_state=0)

transformation = [[0.60834549, -0.63667341], [-0.40887718, 0.85253229]]

X_aniso = np.dot(X, transformation)

clusterDF = pd.DataFrame(data=X_aniso, columns=['ftr1', 'ftr2'])
clusterDF['target'] = y

visualize_cluster_plot(None, clusterDF, 'target', iscenter=False)
```

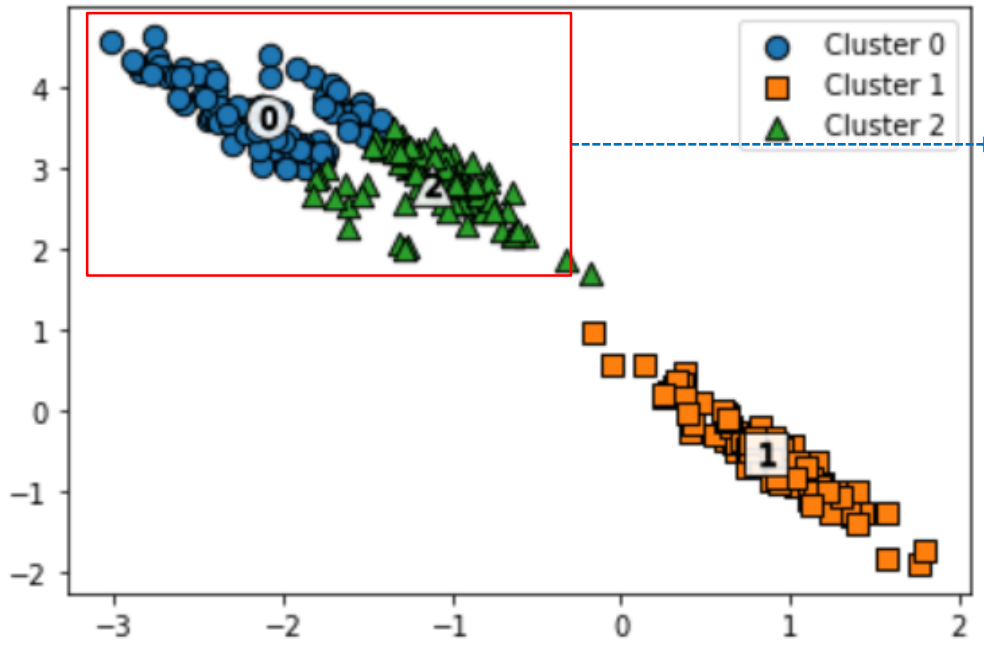
std=0.5는 분산이 작아서 원형으로 데이터가 분포됨



GMM을 이용한 붓꽃 군집화(5)

Kmeans를 이용한 군집화 및 시각화

```
kmeans = KMeans(3, random_state=0)
kmeans_label = kmeans.fit_predict(X_aniso)
clusterDF['kmeans_label'] = kmeans_label

visualize_cluster_plot(kmeans, clusterDF, 'kmeans_label',iscenter=True)
```

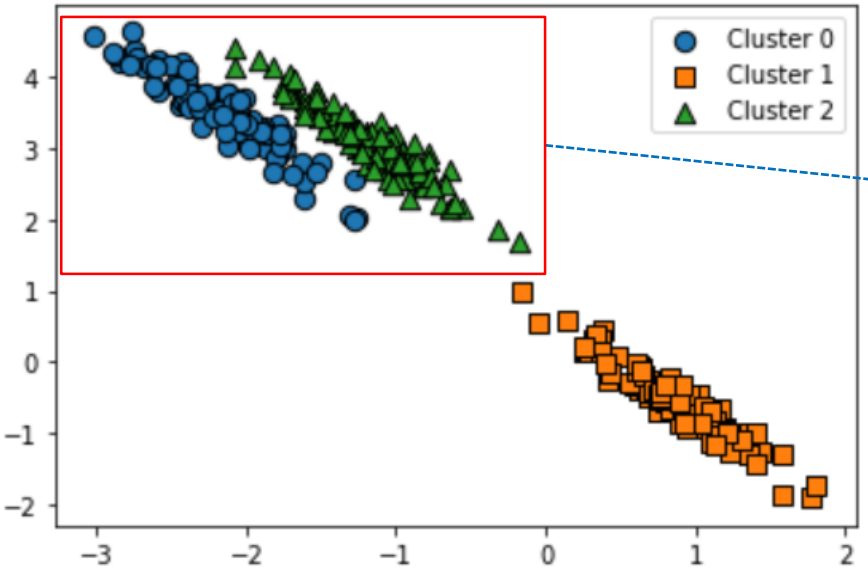


클러스터 0과 클러스터 2개 많이 섞여있다.

Kmeans는 원형이 안닌 길죽한 방향으로 밀집한 데이터의 경우 최적의 군집화가 어렵다.

GMM을 이용한 붓꽃 군집화(6)

```
gmm = GaussianMixture(n_components=3, random_state=0)
gmm_label = gmm.fit(X_aniso).predict(X_aniso)
clusterDF['gmm_label'] = gmm_label


visualize_cluster_plot(gmm, clusterDF, 'gmm_label',iscenter=False)
```



데이터의 분포 방향에 따라서
정확하게 군집화가 되었음.

GMM을 이용한 붓꽃 군집화(7)

```
print('### KMeans Clustering ###')
print(clusterDF.groupby('target')['kmeans_label'].value_counts())
print('₩n### Gaussian Mixture Clustering ###')
print(clusterDF.groupby('target')['gmm_label'].value_counts())
```

```
### KMeans Clustering ###
target  kmeans_label
0       2             73
        0             27
1       1            100
2       0             86
        2             14
Name: kmeans_label, dtype: int64

### Gaussian Mixture Clustering ###
target  gmm_label
0       2            100
1       1            100
2       0            100
Name: gmm_label, dtype: int64
```

Kmeans의 경우 target 값과 label 1만 정확하게 일치함.

GMM의 경우 모든 target 값과 label 값이 정확하게 일치함.

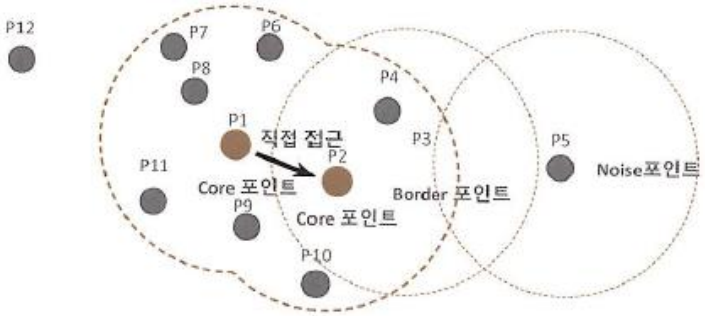
UST 과학기술연합대학원대학교
UNIVERSITY OF SCIENCE & TECHNOLOGY

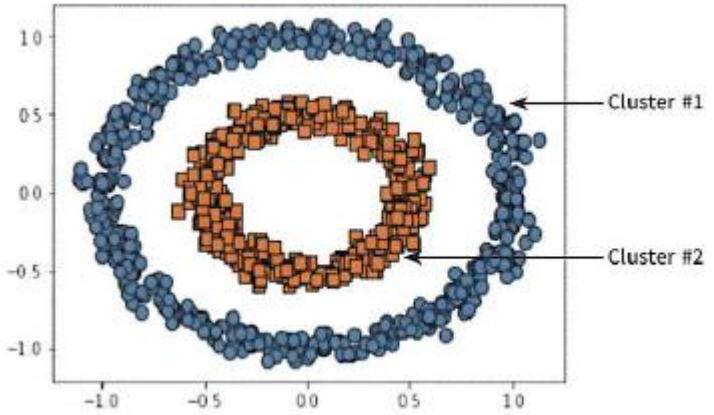
05.
DBSCAN 소개

이홍석 (hsyi@kisti.re.kr)

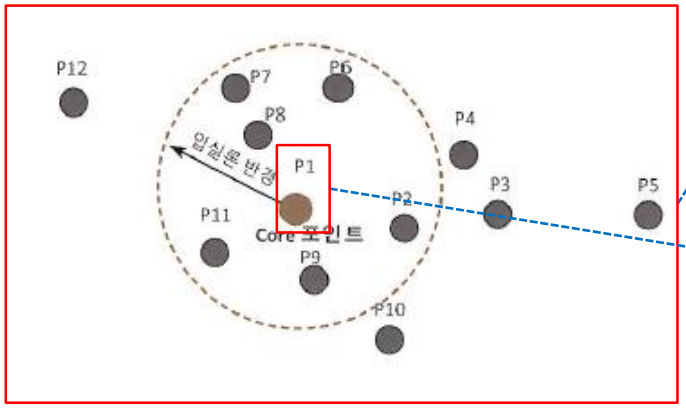
DBSCAN

- DBSCAN 개요
  - DBSCAN(Density Based Spatial Clustering of Application with Noise)
    - 밀도 기반 군집화로 입실론(epsilon) 주변영역
    - 최소 데이터 개수 (min_samples)
    - epsilon 주변 영역의 min_samples를 포함하는 밀도 기준을 충족시키는 데이터인 핵심 포인트를 연결하면서 군집화





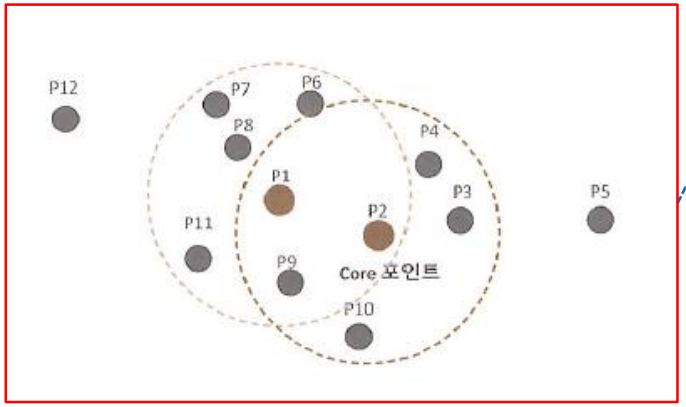
DBSCAN 개요



P1에서 P12까지 12개의 점이 있다면, DBSCAN 군집화 과정은?

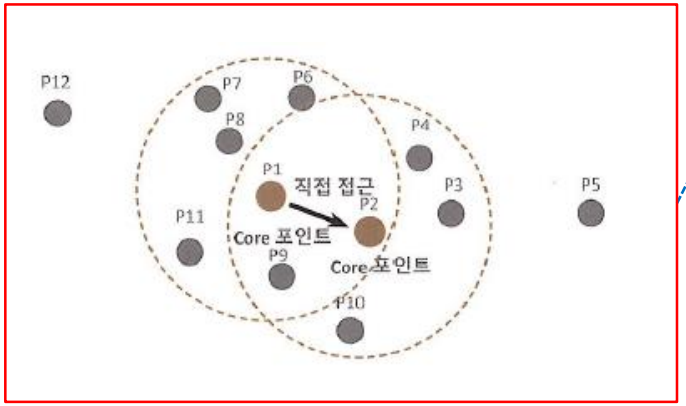
- 임이의 점(P1)에서 특정 입실론 반경 내에 포함될 수 있는 최소 데이터는 자기 자신을 포함하여 6개임

P1은 최소 5개 이상 데이터를 입실론 반경 이내에 포함하므로, P1 데이터는 핵심 포인트임.

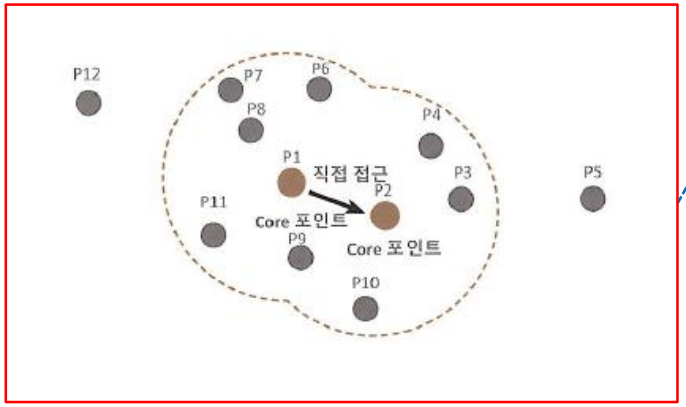


P2 데이터도 자신 포함 6개를 입실론 반경에 포함하므로, P2도 핵심 포인트

DBSCAN 개요

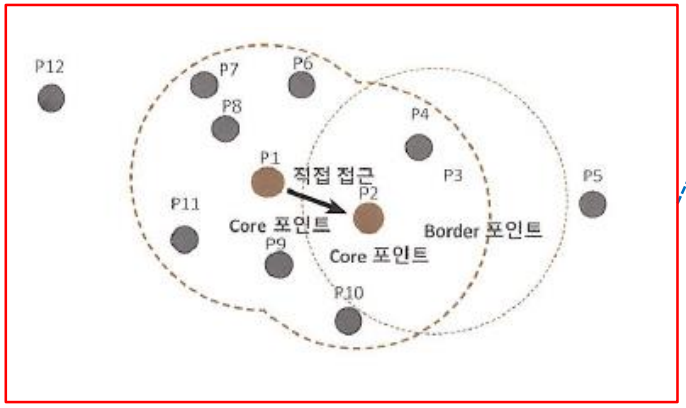


핵심포인트 P1과 이웃의 데이터가 핵심포인트 인 경우 (P2)
- P1에서 P2로 직접 접근이 가능

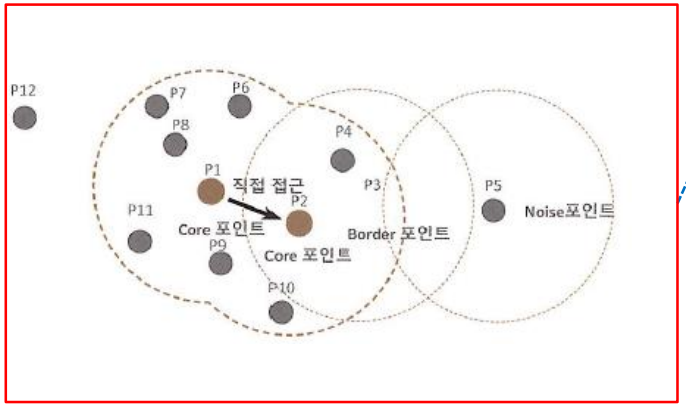


특정 핵심 포인트에서 직접접근이 가능한 다른 핵심포인트를 서로 연결하면서 군집화를 구성.
이러한 점차적 군집 방법을 DBSCAN이라고 함.

DBSCAN 개요



P3는 경계 포인트라고 함. 핵심 포인트는 아님



P5는 노이즈 포인트.

DBSCAN 적용하기 - 붓꽃 (1)

DBSCAN 적용하기 – 붓꽃 데이터 셋

```
from sklearn.datasets import load_iris

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
%matplotlib inline

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

# 보다 편리한 데이타 Handling을 위해 DataFrame으로 변환
irisDF = pd.DataFrame(data=iris.data, columns=feature_names)
irisDF['target'] = iris.target
```

DBSCAN 적용하기 - 붓꽃 (2)

군집 개수는 자동으로 지정됨. 임의의 값을 넣는 것은 의미가 없음.
입실론 (eps) 값과 min_samples 값으로 군집을 자동으로 찾음.

```
from sklearn.cluster import DBSCAN

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)
```

```
target  dbscan_cluster
0        0               49
        -1                1
1        1               46
        -1                4
2        1               42
        -1                8
Name: dbscan_cluster, dtype: int64
```

dbscan 클러스터 -1은 노이즈임
따라서, 붓꽃은 0,1로 dbscan은 군잡화 하고 있음

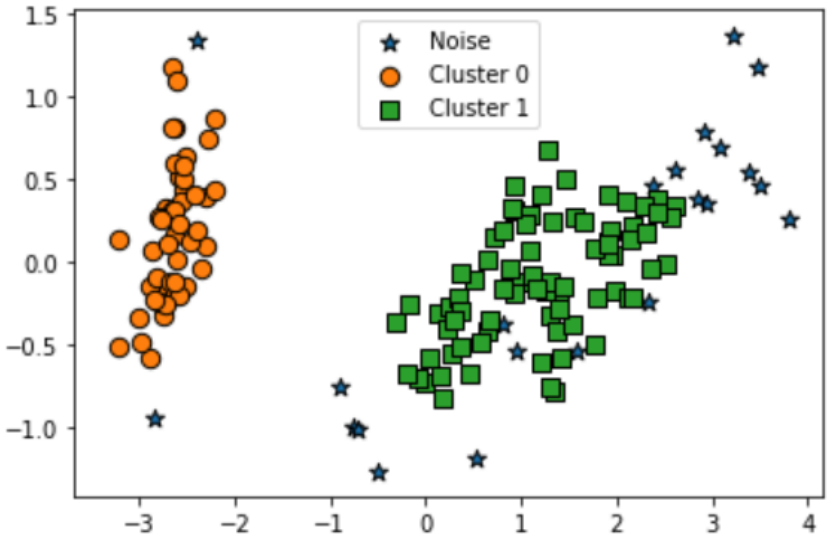
DBSCAN 적용하기 - 붓꽃 (3)

```
### 클러스터 결과를 담은 DataFrame과 사이킷런의 Cluster 객체등을 인자로 받아 클러스터링 결과를 시각화하는 함수
def visualize_cluster_plot(clusterobj, dataframe, label_name, iscenter=True):
    if iscenter :
        centers = clusterobj.cluster_centers_

    unique_labels = np.unique(dataframe[label_name].values)
    markers=['o', 's', '^', 'x', '*']
    isNoise=False

    for label in unique_labels:
        label_cluster = dataframe[dataframe[label_name]==label]
        if label == -1:
            cluster_legend = 'Noise'
            isNoise=True
        else :
            cluster_legend = 'Cluster '+str(label)

        plt.scatter(x=label_cluster['ftr1'], y=label_cluster['ftr2'], s=70,₩
                    edgecolor='k', marker=markers[label], label=cluster_legend)

        if iscenter:
            center_x_y = centers[label]
            plt.scatter(x=center_x_y[0], y=center_x_y[1], s=250, color='white',
                        alpha=0.9, edgecolor='k', marker=markers[label])
            plt.scatter(x=center_x_y[0], y=center_x_y[1], s=70, color='k',₩
                        edgecolor='k', marker='$%d$' % label)
    if isNoise:
        legend_loc='upper center'
    else: legend_loc='upper right'

    plt.legend(loc=legend_loc)
    plt.show()
```

DBSCAN 적용하기 - 붓꽃 (1)

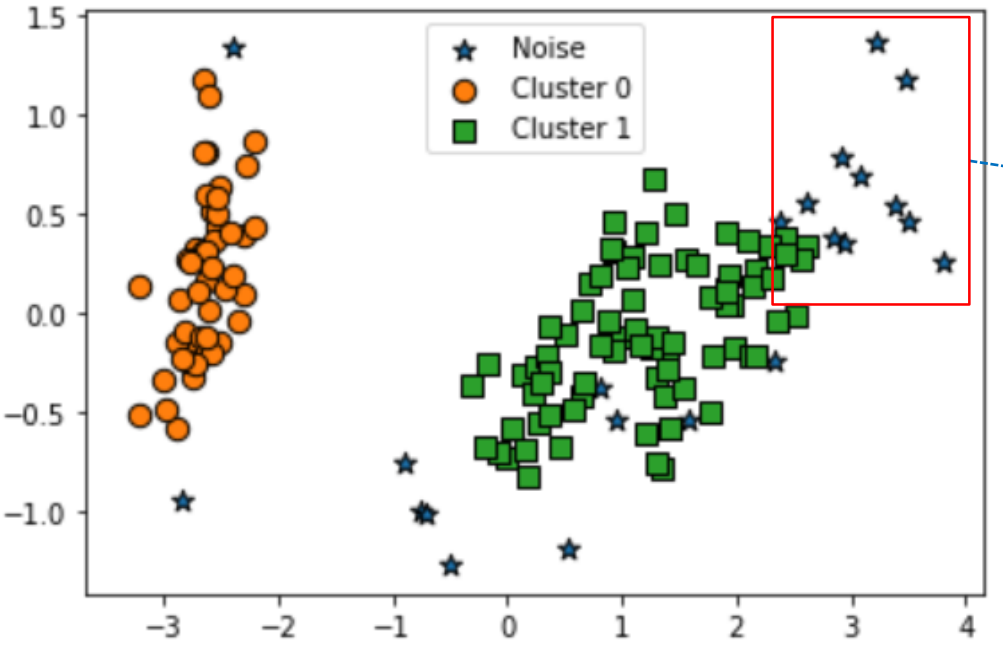
DBSCAN 시각화를 위해 붓꽃을 PCA로 2개의 특성.

```
from sklearn.decomposition import PCA

pca = PCA(n_components=2, random_state=0)
pca_transformed = pca.fit_transform(iris.data)

irisDF['ftr1'] = pca_transformed[:,0]
irisDF['ftr2'] = pca_transformed[:,1]

visualize_cluster_plot(dbscan, irisDF, 'dbscan_cluster', iscenter=False)
```



DBSCAN 적용하기 - 붓꽃 (4)

```
visualize_cluster_plot(dbscan, irisDF, 'dbscan_cluster', iscenter=False)
```



별로 표시된 데이터는
노이즈.

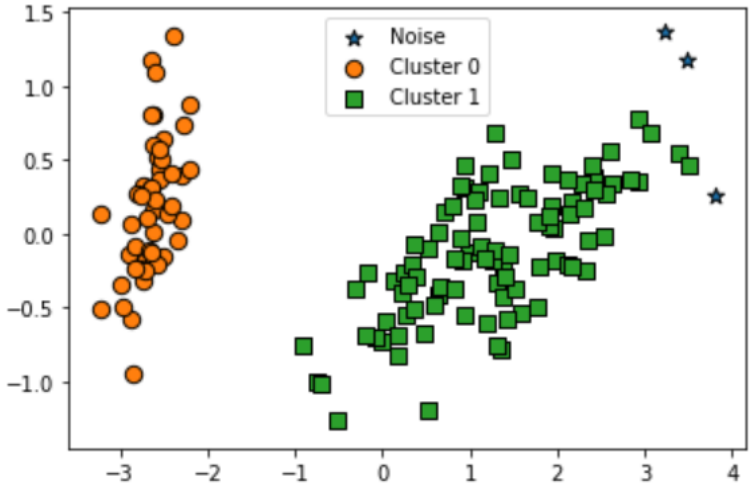
DBSCAN 적용하기 - 붓꽃 (5)

```
from sklearn.cluster import DBSCAN

dbscan = DBSCAN(eps=0.8, 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)

visualize_cluster_plot(dbscan, irisDF, 'dbscan_cluster', iscenter=False)
```

DBSCAN의 입실론 크기:
eps=0.6에서 0.8변경

```
target  dbscan_cluster
0        0               50
1        1               50
2        1               47
        -1                3
Name: dbscan_cluster, dtype: int64
```

반경이 커져서, 노이즈는 줄어듬



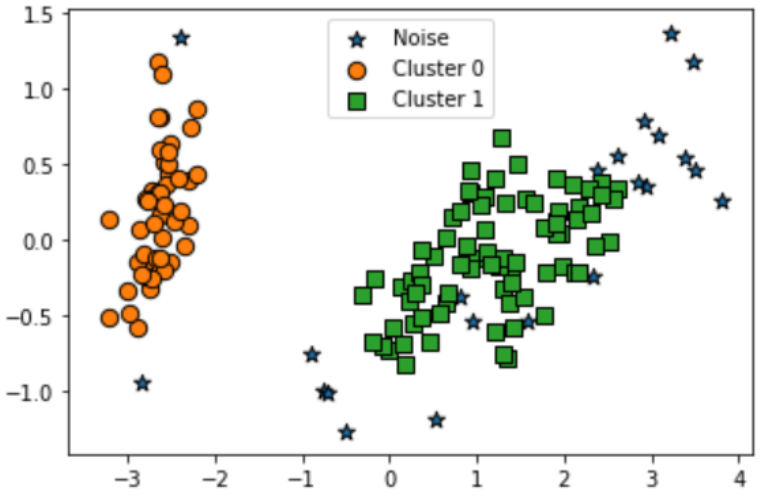
DBSCAN 적용하기 - 붓꽃 (6)

DBSCAN의 입실론 크기: min_samples 변경하기

```
dbscan = DBSCAN(eps=0.6, min_samples=16, 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)
visualize_cluster_plot(dbscan, irisDF, 'dbscan_cluster', iscenter=False)
```

min_samples=16, 노이즈는 기존 보다 많이 증가한다.

```
target  dbscan_cluster
0        0                48
        -1                 2
1        1                44
        -1                 6
2        1                36
        -1                14
Name: dbscan_cluster, dtype: int64
```

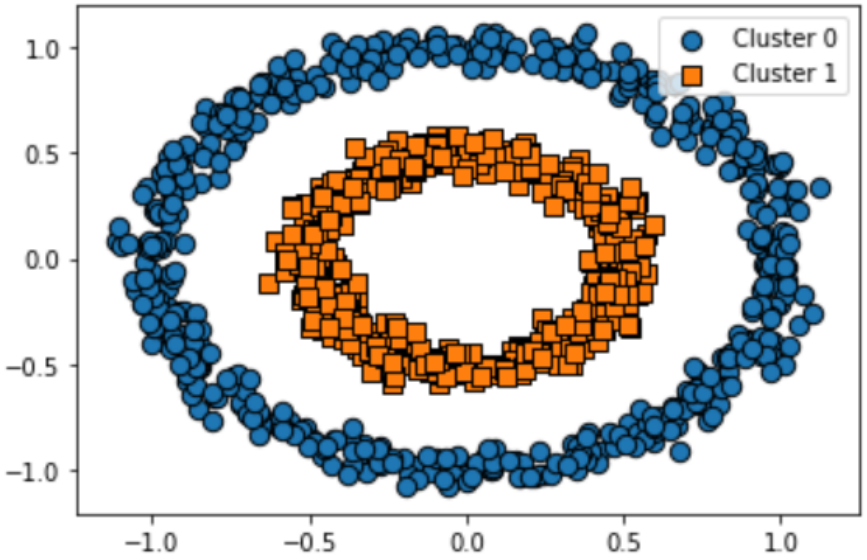


DBSCAN 적용하기: 방울 (1)

DBSCAN 적용하기 – make_circles() 데이터 세트

```
from sklearn.datasets import make_circles

X, y = make_circles(n_samples=1000, shuffle=True, noise=0.05, random_state=0, factor=0.5)
clusterDF = pd.DataFrame(data=X, columns=['ftr1', 'ftr2'])
clusterDF['target'] = y

visualize_cluster_plot(None, clusterDF, 'target', iscenter=False)
```

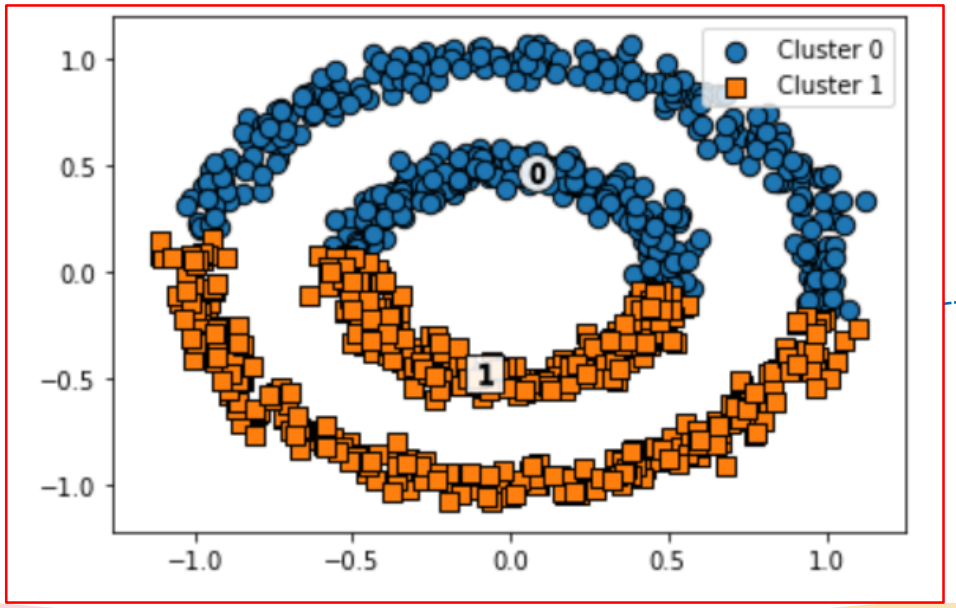
내부 원과 외부 원의 scale 비율



KMeans 적용하기:예제(2)

Kmeans로 방울 데이터 분류.

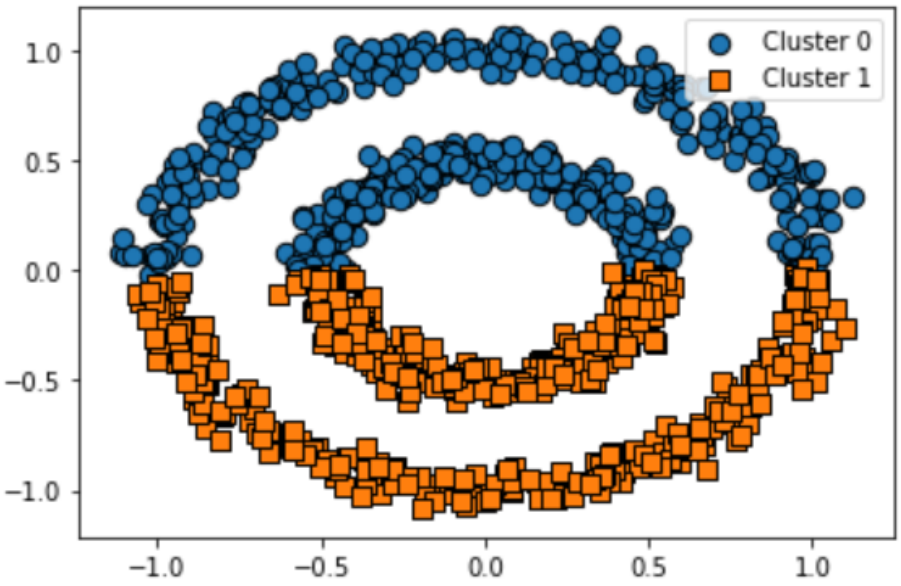
```
# KMeans로 make_circles( ) 데이터 셋을 클러스터링 수행.
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=2, max_iter=1000, random_state=0)
kmeans_labels = kmeans.fit_predict(X)
clusterDF['kmeans_cluster'] = kmeans_labels

visualize_cluster_plot(kmeans, clusterDF, 'kmeans_cluster', iscenter=True)
```



원의 위와 아래로 분류

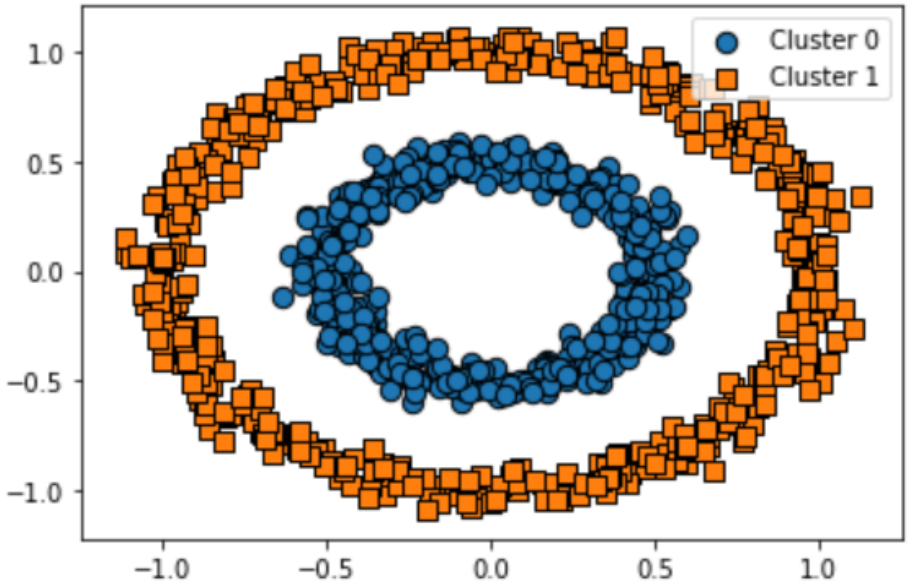
GMM 적용하기:방울 예제(3)

```
# GMM으로 make_circles( ) 데이터 셋을 클러스터링 수행.
from sklearn.mixture import GaussianMixture

gmm = GaussianMixture(n_components=2, random_state=0)
gmm_label = gmm.fit(X).predict(X)
clusterDF['gmm_cluster'] = gmm_label

visualize_cluster_plot(gmm, clusterDF, 'gmm_cluster', iscenter=False)
```



DBSCAN 적용하기:예제(4)

```
# DBSCAN으로 make_circles( ) 데이터 셋을 클러스터링 수행.
from sklearn.cluster import DBSCAN

dbscan = DBSCAN(eps=0.2, min_samples=10, metric='euclidean')
dbscan_labels = dbscan.fit_predict(X)
clusterDF['dbscan_cluster'] = dbscan_labels

visualize_cluster_plot(dbscan, clusterDF, 'dbscan_cluster', iscenter=False)
```



UST 과학기술연합대학원대학교
UNIVERSITY OF SCIENCE & TECHNOLOGY

Thank You!

www.ust.ac.kr