



# Aichemist Session

CHAP 08 (군집화)

# CONTENTS

## 챕터 제목

1. K-평균 알고리즘 이해
2. 군집 평가
3. 평균 이동
4. GMM
5. DBSCAN

---

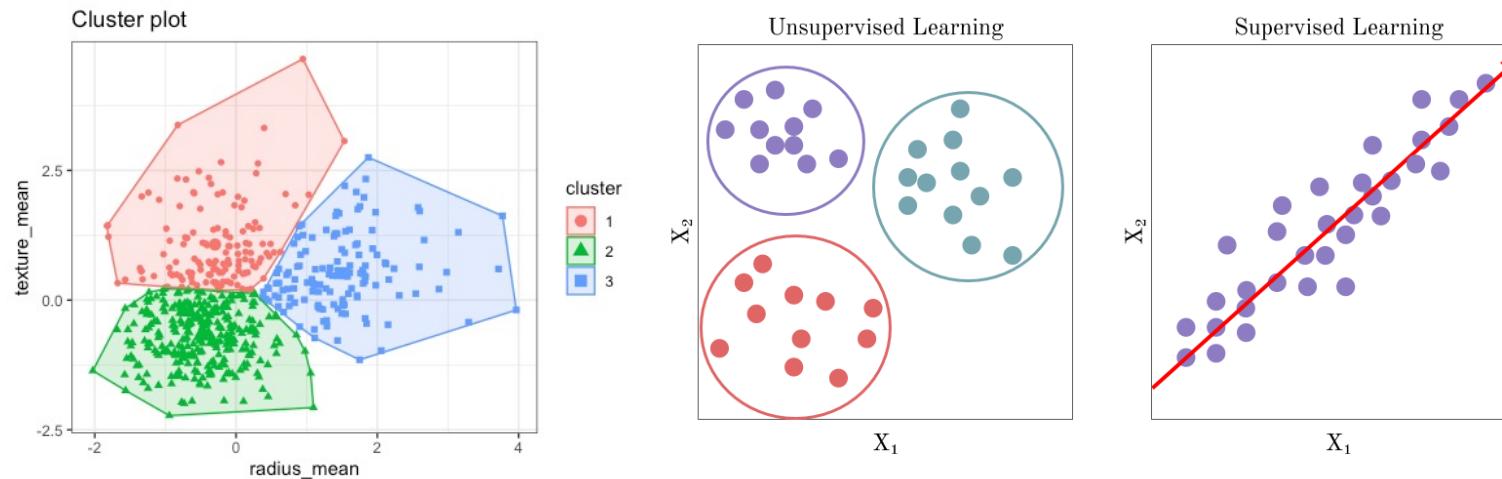


01.

## K-평균 알고리즘 이해

# 군집화란 ?

- 머신러닝에서 군집화는 **비지도학습**의 한 종류로, 유사한 특성을 가진 데이터를 같은 그룹(즉, 군집)으로 묶는 과정을 의미
- 군집화 알고리즘은 데이터의 특성을 분석하여 서로 유사한 데이터끼리 그룹화하고, 이렇게 생성된 그룹들 사이의 차이를 최대화하는 방식으로 작동
- 군집화는 레이블이 없는 데이터에 대한 이해를 돋거나, 데이터의 이상치를 탐지하고, 추천 시스템 등 다양한 분야에서 사용



# k-평균 알고리즘

- 군집 중심점이라는 특정한 임의 지점을 선택해 해당 중심에 가장 가까운 포인트들을 선택하는 군집화 기법
- 선택된 포인트의 **평균위치** 으로 이동하고 이동된 중심점에서 다시 가까운 포인트 선택, 다시 중심점을 평균지점으로 이동하는  
프로세스 반복 수행

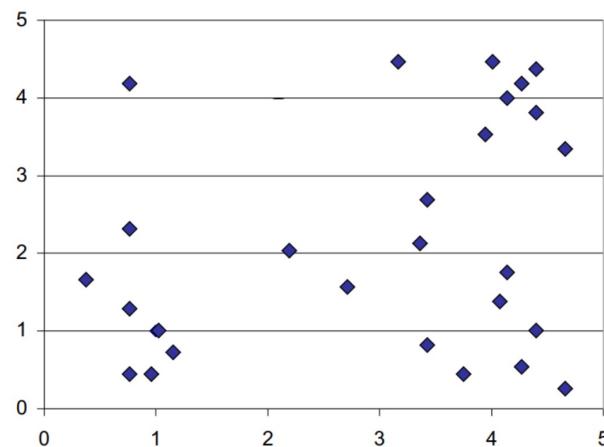
---

## Algorithm 1 $k$ -means algorithm

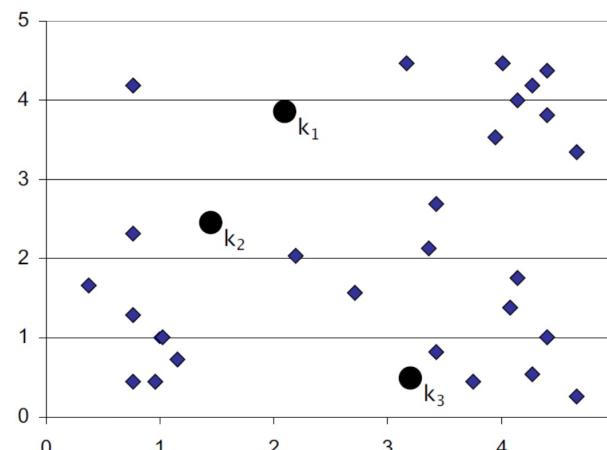
---

- 1: Specify the number  $k$  of clusters to assign.
  - 2: Randomly initialize  $k$  centroids.
  - 3: **repeat**
  - 4:   **expectation:** Assign each point to its closest centroid.
  - 5:   **maximization:** Compute the new centroid (mean) of each cluster.
  - 6: **until** The centroid positions do not change.
-

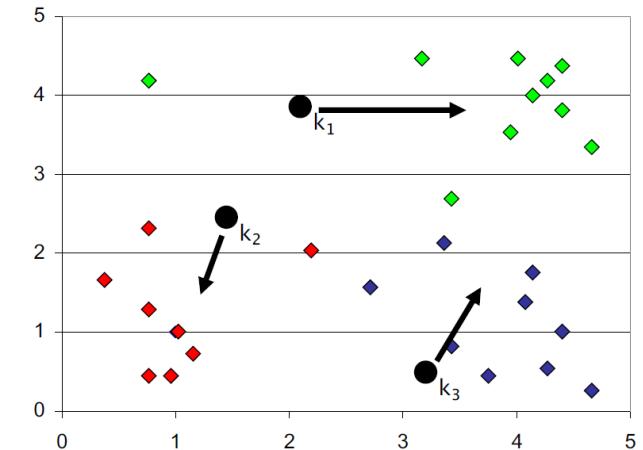
# k-평균 알고리즘



1. 몇 개의 뎅어리로 clustering할지 정한다  
Ex) 3개의 뎅어리로 clustering 수행

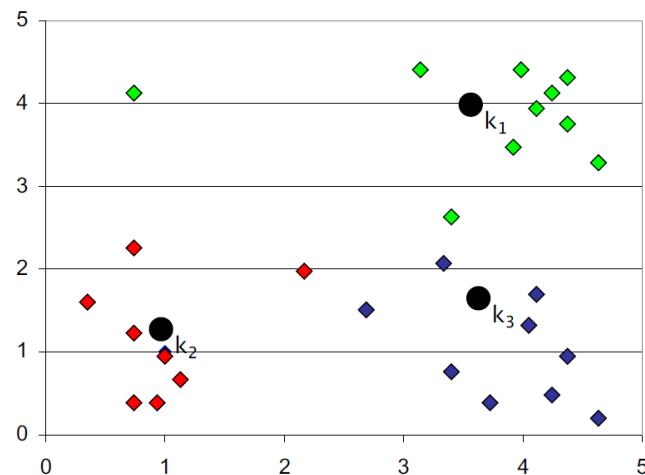


2. 1에서 정한 개수만큼 중심점을 정한다  
(자신이 원하는 아무 값으로 중심점을 정하기)

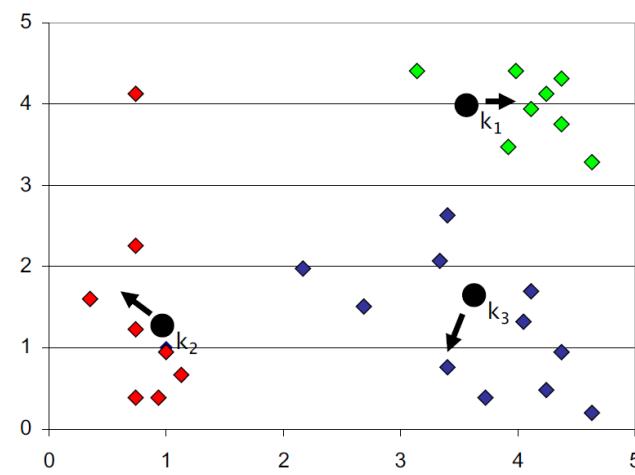


3. 각 점에 대해 가장 가까운 centroid를 정한다

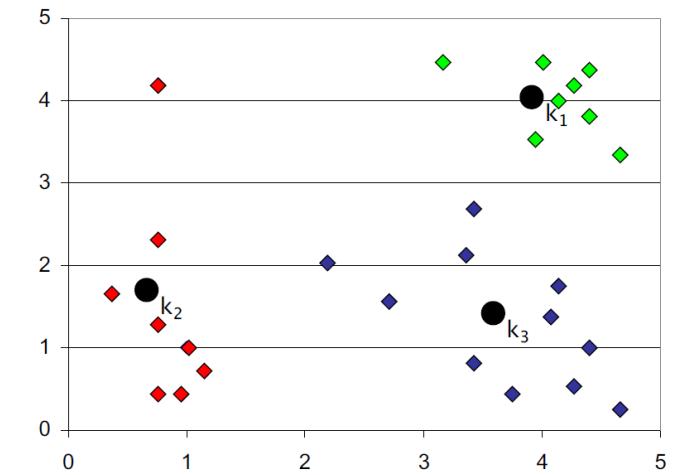
# k-평균 알고리즘



4. 이제 각 매핑된 점들을 바탕으로 하여  
centroid를 이동한다



5. 3-4의 과정을 더이상 새로 매핑되지 않  
을 때까지 반복한다



6. 새로 매핑된 점들을 바탕으로 다시  
centroid를 이동한다

위의 과정을 더이상 이동이 없을 때까지 반복

# k-평균 알고리즘

장점	단점
일반적인 군집화에서 가장 많이 활용됨	거리 기반 알고리즘 속성의 개수가 많을 경우 군집화의 정확도가 떨어짐 (PCA 차원 감소를 적용해야 할 수 있음)
알고리즘이 쉽고 간결	반복을 수행하는데, 반복 횟수가 많을 경우 수행시간이 매우 느려짐
	몇 개의 군집을 선택해야 할지 가이드하기 어려움

# 사이킷런 Kmeans 클래스 소개

kmean 주요 파라미터	
<code>n_clusters</code>	군집화할 개수 (군집 중심점 개수)
<code>init</code>	초기 군집 중심점 좌표를 설정할 방식 보통 임의의 중심을 설정 X 일반적으로 k-means++ 방식으로 최초 설정
<code>Max_iter</code>	최대 반복 횟수 이 횟수 이전에 모든 데이터의 중심점 이동이 없으면 종료

kmean 객체 주요 속성	
<code>Labels_</code>	각 데이터 포인트가 속한 군집 중심점 레이블
<code>Cluster_centers_</code>	각 군집 중심점 좌표(shape: [군집 개수, 피처 개수]) 이를 통해 군집 중심점 좌표가 어디인지 시각화 가능

# K-평균을 이용한 붓꽃 데이터 세트 군집화

꽃받침, 꽃잎의 길이에 따라 각 데이터의 군집화가 어떻게 결정되는지 확인하고, 분류 값과 비교

## ① 데이터 로드

```
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()
# 보다 편리한 데이터 Handling을 위해 DataFrame으로 변환
irisDF = pd.DataFrame(data=iris.data, columns=['sepal_length','sepal_width','petal_length','petal_width'])
irisDF.head(3)
```

	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

# K-평균을 이용한 붓꽃 데이터 세트 군집화

## ② 군집화

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

```
|: KMeans(n_clusters=3, random_state=0)
```

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

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

target cluster

target cluster

1 0 48

- 2

2 0 14

- target = 실제 분류값

- Cluster = 군집화 분류값

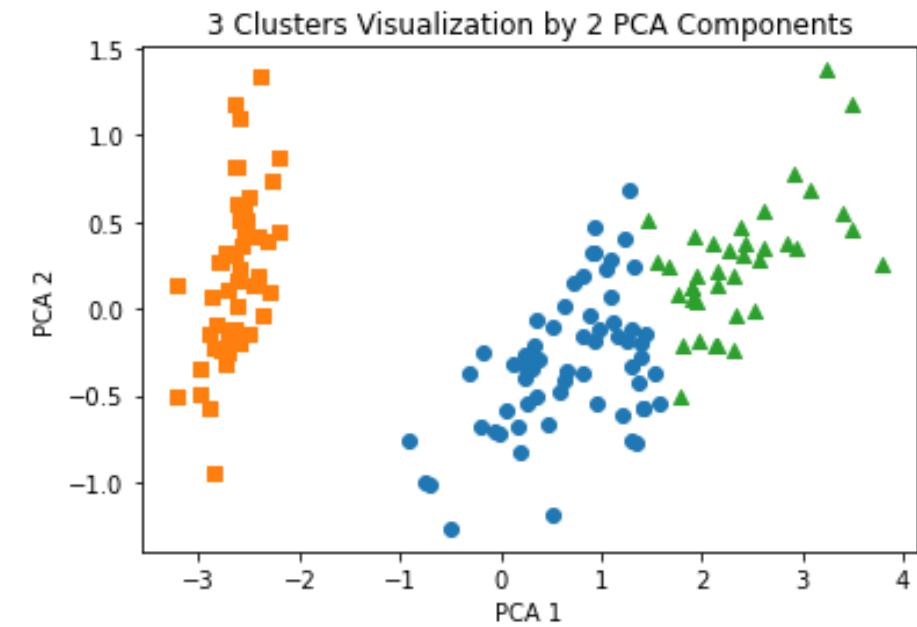
2 36

# K-평균을 이용한 붓꽃 데이터 세트 군집화

## ③ 군집화 시작화

PCA 이용해 4개의 붓꽃 속성을 2개로 차원 축소한 뒤에 X좌표, Y좌표로 개별 데이터를 표현

```
from sklearn.decomposition import PCA  
  
pca = PCA(n_components=2)  
pca_transformed = pca.fit_transform(iris.data)  
  
irisDF['pca_x'] = pca_transformed[:,0]  
irisDF['pca_y'] = pca_transformed[:,1]  
irisDF.head(3)
```



# 군집화 알고리즘을 위한 데이터 생성

- 군집화용 데이터 생성기
- 여러 개의 클래스에 해당되는 데이터 세트 만듦
- 하나의 클래스에 여러 개의 군집이 분포될 수 있게 하는 데이터를 생성할 수 있음

<code>make_blobs</code>	- 개별 군집의 중심점과 표준편차 제어 기능 추가 - 피처 데이터 세트와 타깃 데이터 세트가 튜플 형태로 반환
<code>make_classification()</code>	- 노이즈를 포함한 데이터를 만드는 데 유용하게 사용

# 군집화 알고리즘을 위한 데이터 생성

Make\_blobs()로 2개 피처가 3개 군집화 기반 분포도  
를 가진 데이터 세트와 타깃 데이터 세트 반환

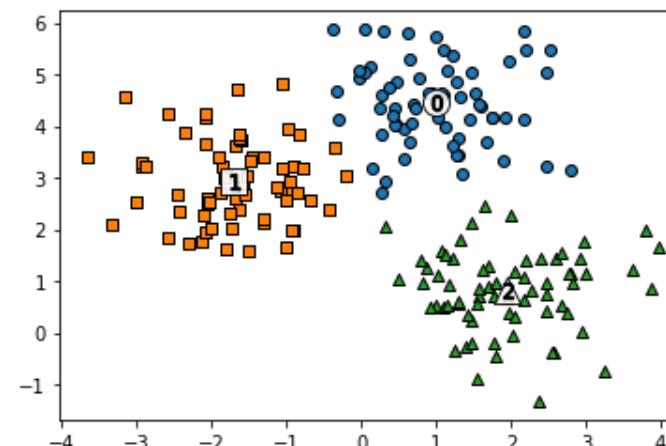
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
%matplotlib inline

X, y = make_blobs(n_samples=200, n_features=2, centers=3, cluster_std=0.8, random_state=0)
print(X.shape, y.shape)

# y target 값의 분포를 확인
unique, counts = np.unique(y, return_counts=True)
print(unique, counts)
```

(200, 2) (200,)  
[0 1 2] [67 67 66]

Cluster\_centers\_ 속성은 개별 군집의 중심 위치  
좌표를 나타냄



---



02.

군집 평가

# 군집 평가

- 군집화 vs 분류

군집화 알고리즘은 분류와 다름. 군집화는 비지도 학습이며 군집화에서 동일한 분류 값에 속하더라도 그 안에서 더 세분화 된 군집화 발견 가능

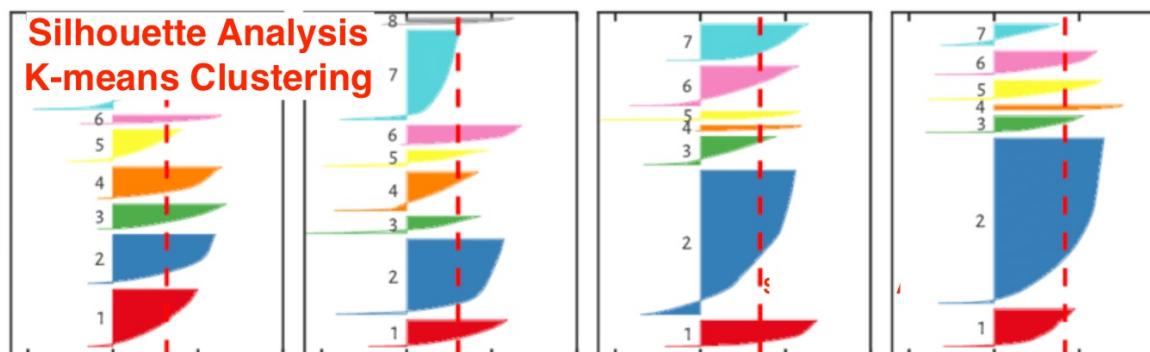
- 실루엣 분석

군집화 성능을 평가하는 대표적인 방법

각 군집 간의 거리가 얼마나 효율적으로 분리돼 있는지를 나타냄

실루엣 계수를 기반으로 평가

- 실루엣 계수 : -1과 1 사이
- 1로 가까워질수록 근처의 군집과 멀리 떨어짐.
- 0에 가까울 수록 근처의 군집과 가까움



# 실루엣 계수

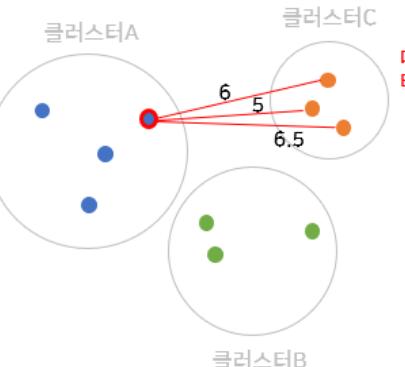
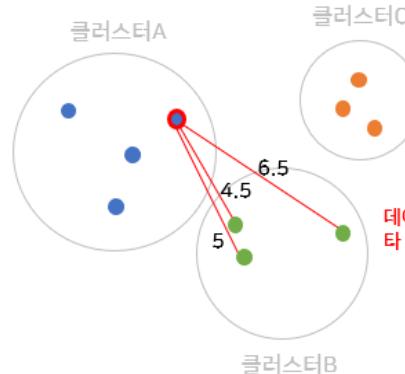
- 실루엣 계수

①



$a(i)$  : 데이터 포인트 i가 속한 클러스터 내 데이터 포인트들과 거리 평균

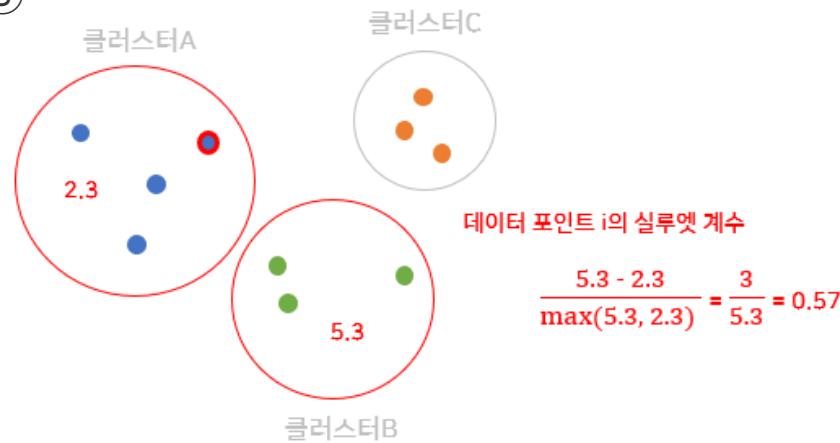
②



$d(i, C)$  : 데이터 포인트 i가 속하지 않은 클러스터 C 의 데이터 포인트들과 거리 평균

# 실루엣 계수

③



$b(i) : d(i, C)$ 의 최솟값

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}, \quad -1 \leq s(i) \leq 1$$

$i$ 번째 데이터 포인트 실루엣 계수 값  $s(i)$

# 붓꽃 데이터 세트를 이용한 군집 평가

1번 군집 실루엣 계수 0.8  
정도로 높은 점수

```
from sklearn.preprocessing import scale
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
# 실루엣 분석 metric 값을 구하기 위한 API 추가
from sklearn.metrics import silhouette_samples, silhouette_score
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)
kmeans = KMeans(n_clusters=3, init='k-means++', max_iter=300, random_state=0).fit(iris)

irisDF['cluster'] = kmeans.labels_

# iris 의 모든 개별 데이터에 실루엣 계수값을 구함.
score_samples = silhouette_samples(iris.data, irisDF['cluster'])
print('silhouette_samples() return 값의 shape', score_samples.shape)

# irisDF에 실루엣 계수 컬럼 추가
irisDF['silhouette_coeff'] = score_samples

# 모든 데이터의 평균 실루엣 계수값을 구함.
average_score = silhouette_score(iris.data, irisDF['cluster'])
print('붓꽃 데이터셋 Silhouette Analysis Score:{0:.3f}'.format(average_score))

irisDF.head(3)

silhouette_samples() return 값의 shape (150,)
붓꽃 데이터셋 Silhouette Analysis Score:0.553
```

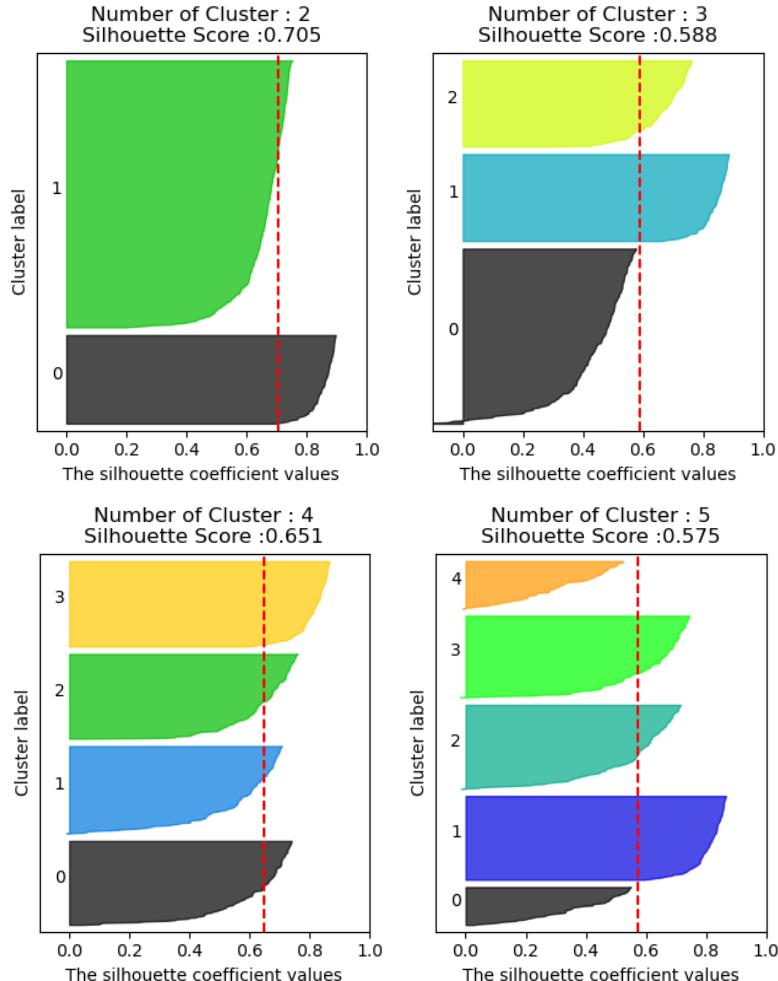
	sepal_length	sepal_width	petal_length	petal_width	cluster	silhouette_coeff
0	5.1	3.5	1.4	0.2	1	0.852955
1	4.9	3.0	1.4	0.2	1	0.815495
2	4.7	3.2	1.3	0.2	1	0.829315

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

```
cluster
0    0.417320
1    0.798140
2    0.451105
Name: silhouette_coeff, dtype: float64
```

다른 군집의 실루엣 계수

# 군집별 평균 실루엣 계수의 시각화를 통한 군집 개수 최적화 방법



## 1. 군집의 개수가 2개일 때

1번 군집의 모든 데이터는 평균 실루엣 계수값이 이상이지만, 2번 군집의 경우는 평균보다 적은 데이터 값이 매우 많음.

## 2. 군집의 개수가 3개 일 경우

0번의 경우 모두 평균보다 평균 실루엣 계수가 낮음

## 3. 군집이 4개인 경우

개별 군집의 평균 실루엣 계수값이 비교적 균일하게 위치

\*k-평균 군집화는 직관적으로 이해하기 쉽지만 데이터 간 거리를 반복적으로 계산해야 되므로 데이터 양이 늘어나면 수행시간이 크게 늘어남.

---

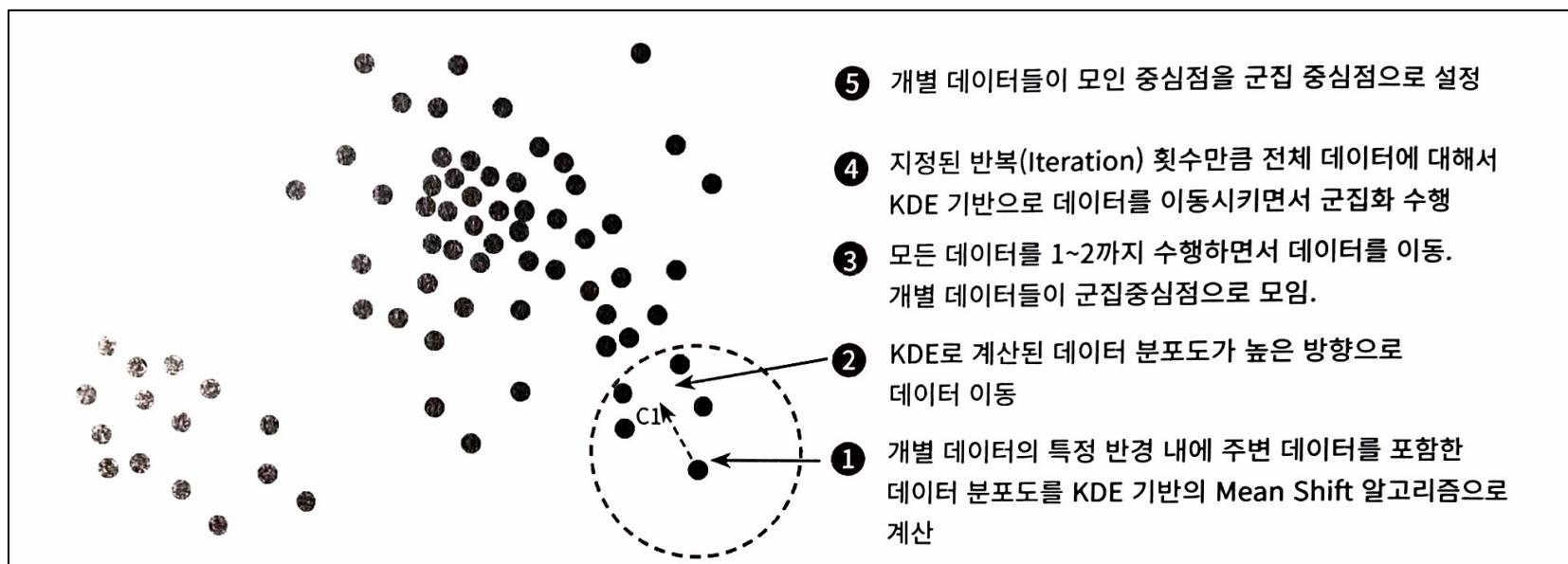


03.

평균 이동

# 평균 이동

- 평균 이동(Mean Shift) 알고리즘은 데이터 포인트들의 밀도가 가장 높은 곳, 즉 데이터 포인트들이 가장 밀집되어 있는 곳을 찾아가는 방식으로 군집화를 수행하는 알고리즘
- 평균 이동 알고리즘은 각 데이터 포인트에서 주어진 범위 내의 데이터 포인트들의 중심(평균)을 계산하고, 이 평균 위치로 데이터 포인트를 이동함. 이 과정을 모든 데이터 포인트에 대해 반복하면서 데이터 포인트들이 모여 있는 중심을 찾아냅니다.
- 장점은 군집의 개수를 미리 지정할 필요가 없다는 것



# KDE(Kernel Density Estimation)

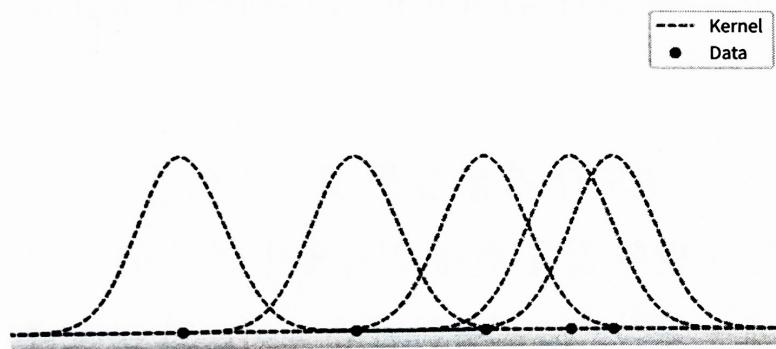
확률 밀도 함수(Probability Density Function, PDF) : 확률 변수의 분포를 나타내는 함수

KDE : kernel 함수를 통해 어떤 변수의 확률 밀도 함수를 추정하는 대표적인 방법

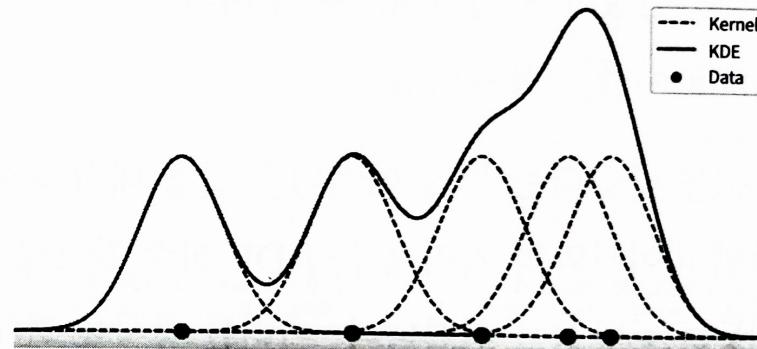
개별 관측 데이터에 커널 함수를 적용한 값을 모두 더한 후 데이터 건수로 나눠 확률 밀도 함수 추정

대표적인 커널 함수로 가우시안 함수(정규분포) 사용

개별 관측 데이터에 가우시안 커널 함수 적용



가우시안 커널 함수 적용 후 합산



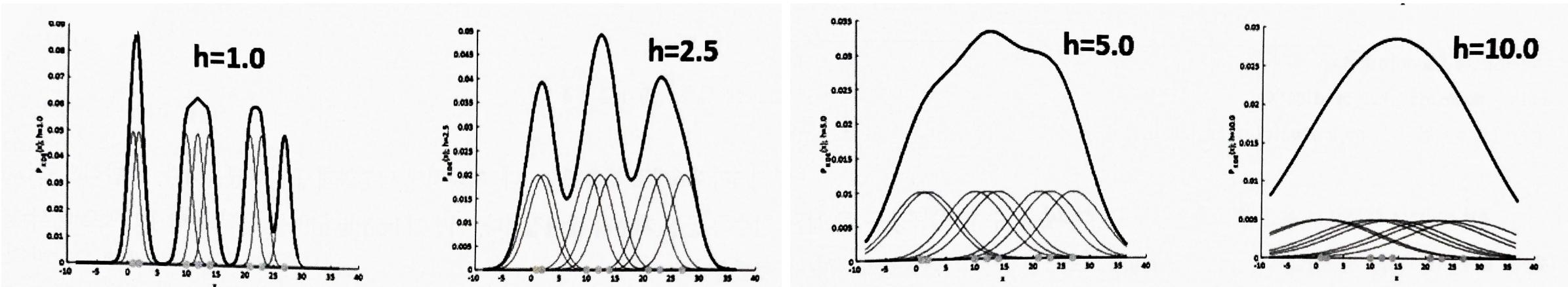
# KDE(Kernel Density Estimation)

대여족 : KDE 형태를 부드러운 형태로 평활화하는데 적용 → 확률 밀도 추정 성능 크게 좌우

$$\text{KDE} = \frac{1}{n} \sum_{i=1}^n K_h(x - x_i) = \frac{1}{nh} \sum_{i=1}^n K\left(\frac{x - x_i}{h}\right)$$

작은  $h$ 값 > 좁고 뾰족한 KDE > 변동성이 큰 방식으로 추정 > 가능성

큰  $h$ 값 > 과도하게 평활화된 KDE > 지나치게 단순화된 방식으로 추정 > 가능성



# 사이킷런 MeanShift 클래스

```
import numpy as np
from sklearn.datasets import make_blobs
from sklearn.cluster import MeanShift
```

표준편차가 0.7인 3개 군집의 데이터 생성

```
X, y = make_blobs(n_samples=200, n_features=2, centers=3,
                    cluster_std=0.7, random_state=0)
```

```
meanshift = MeanShift(bandwidth=0.8)
```

KDE 대역폭 h = bandwidth 주요 파라미터  
print('cluster labels 유형:', np.unique(cluster\_labels))

cluster labels 유형: [0 1 2 3 4 5]

```
meanshift = MeanShift(bandwidth=1)
cluster_labels = meanshift.fit_predict(X)
print('cluster labels 유형:', np.unique(cluster_labels))
```

cluster labels 유형: [0 1 2]

```
from sklearn.cluster import estimate_bandwidth
```

```
bandwidth = estimate_bandwidth(X)
print('bandwidth 값:', round(bandwidth, 3))
```

bandwidth 값: 1.816

Estimate\_bandwidth() : 최적화된 bandwidth 찾는 함수  
파라미터로 피처 데이터 세트 입력

# 사이킷런 MeanShift 클래스

```
import pandas as pd

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

# estimate_bandwidth()로 최적의 bandwidth 계산
best_bandwidth = estimate_bandwidth(X)

meanshift = MeanShift(bandwidth=best_bandwidth)
cluster_labels = meanshift.fit_predict(X)
print('cluster labels 유형:', np.unique(cluster_labels))

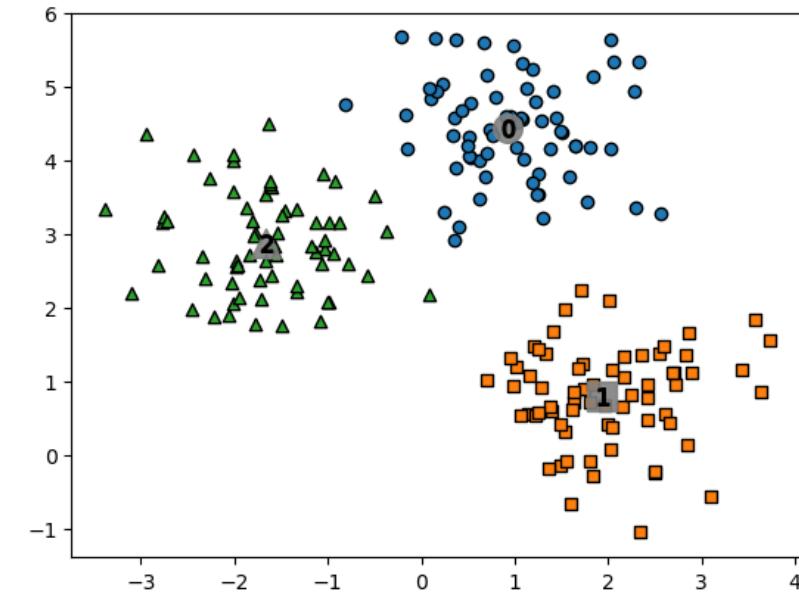
import matplotlib.pyplot as plt
%matplotlib inline

clusterDF['meanshift_label'] = cluster_labels
centers = meanshift.cluster_centers_
unique_labels = np.unique(cluster_labels)
markers=['o', 's', '^', 'x', '*']

for label in unique_labels:
    label_cluster = clusterDF[clusterDF['meanshift_label']==label]
    center_x_y = centers[label]
    # 굽침별로 다른 마커로 산점도 적용
    plt.scatter(x=label_cluster['ftr1'], y=label_cluster['ftr2'], edgecolor='k', marker=markers[label] )

    # 굽침별 중심 표현
    plt.scatter(x=center_x_y[0], y=center_x_y[1], s=200, color='gray', alpha=0.9, marker=markers[label])
    plt.scatter(x=center_x_y[0], y=center_x_y[1], s=70, color='k', edgecolor='k', marker='$_d$' % label)

plt.show()
```



```
print(clusterDF.groupby('target')['meanshift_label'].value_counts())

target  meanshift_label
0        0                  67
1        1                  67
2        2                  66
Name: count, dtype: int64
```

## 평균 이동 장단점

장점	단점
<ul style="list-style-type: none"><li>데이터 세트 형태를 특정 형태로 가정하거나, 특정 분포도 기반 모델로 가정하지 않음</li><li>이상치의 영향력이 크지 않음 미리 군집 개수를 정할 필요 없음</li></ul>	<ul style="list-style-type: none"><li>알고리즘 수행시간이 오래 걸림</li><li>대역폭에 따라 군집화 성능이 좌우됨</li></ul>

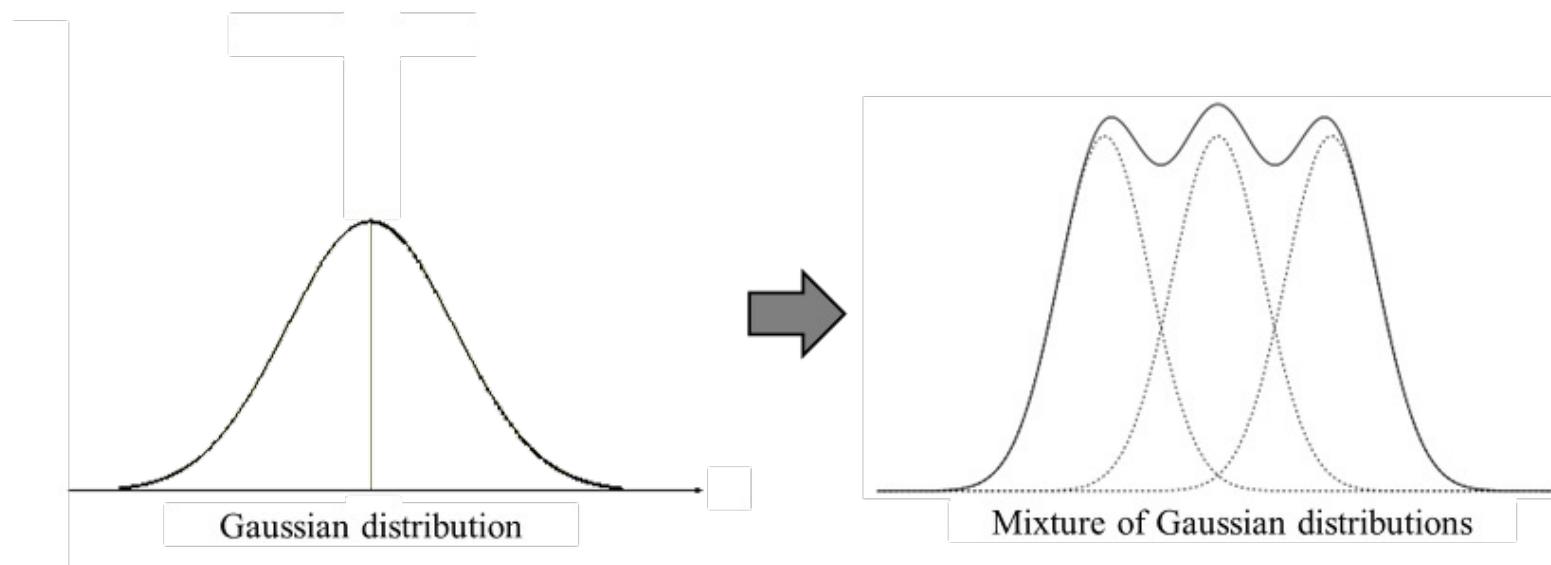
04.

GMM

# GMM(Gaussian Mixture Model)

군집화를 적용하고자 하는 데이터가 여러개의 가우시안 분포를 가진 데이터 집합들이 섞여서 생성된 것이라는 가정하에 군집화를 수행하는 방식

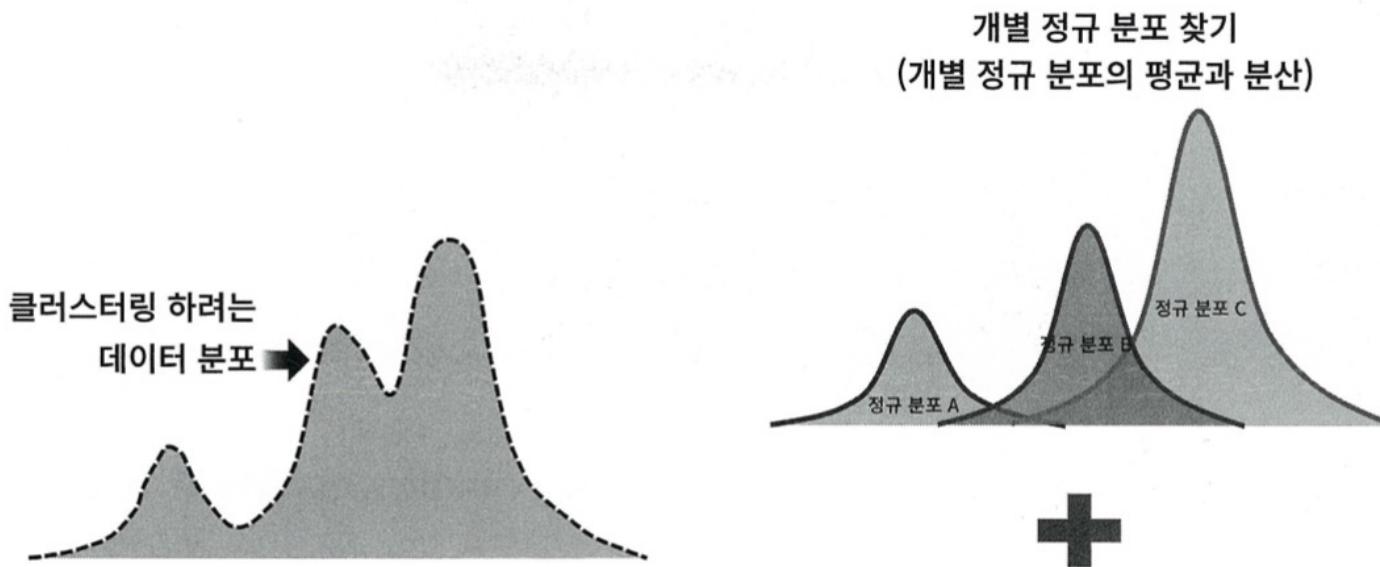
섞인 데이터 분포에서 개별 유형의 가우시안 추출 → 개별 데이터가 이 중 어떤 정규 분포에 속하는지 결정



# 모수 추정

## 모수 추정

- (1) 개별 정규 분포의 평균과 분산을 추정에 값 반환
- (2) 각 데이터가 어떤 정규 분포에 해당되는지의 확률을 추정에 값 반환



데이터가 특정 정규 분포에 해당될 확률 구하기

# 사이킷런 GaussianMixture 클래스

```
from sklearn.mixture import GaussianMixture    n_components : 가우시안 혼합 모델 총 개수(군집 개수 설정)
gmm = GaussianMixture(n_components=3, random_state=0).fit(iris.data)
gmm_cluster_labels = gmm.predict(iris.data)

# 클러스터링 결과를 irisDF 의 'gmm_cluster' 컬럼명으로 저장
irisDF['gmm_cluster'] = gmm_cluster_labels
irisDF['target'] = iris.target

# target 값에 따라서 gmm_cluster 값이 어떻게 매핑되었는지 확인.
iris_result = irisDF.groupby(['target'])['gmm_cluster'].value_counts()
print(iris_result)
```

```
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	count
0	1	50
1	0	48
	2	2
2	2	36
	0	14

# GMM과 K-Means 비교

Kmeans는 데이터가 타원형인 데이터에 대해 군집화를 잘 수행하지 못함

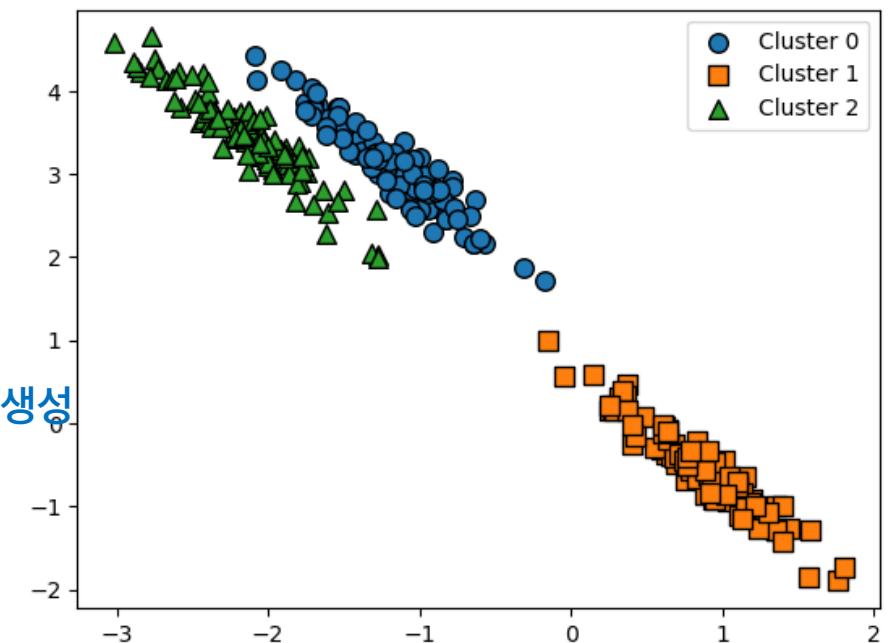
→ 타원형 데이터에 대해 GMM과 Kmeans 결과 비교

```
from sklearn.datasets import make_blobs

# make_blobs()로 300개의 데이터 셋, 3개의 cluster 셋, cluster_std=0.5을 만듬.
X, y = make_blobs(n_samples=300, n_features=2, centers=3, cluster_std=0.5, random_state=0)

# 길게 늘어난 타원형의 데이터 셋을 생성하기 위해 변환함.
transformation = [[0.60834549, -0.63667341], [-0.40887718, 0.85253229]]
X_aniso = np.dot(X, transformation)
# feature 데이터 셋과 make_blobs()의 y 결과 값을 DataFrame으로 저장
clusterDF = pd.DataFrame(data=X_aniso, columns=['ftr1', 'ftr2'])
clusterDF['target'] = y
# 생성된 데이터 셋을 target 별로 다른 marker로 표시하여 시각화 함.
visualize_cluster_plot(None, clusterDF, 'target', iscenter=False)
```

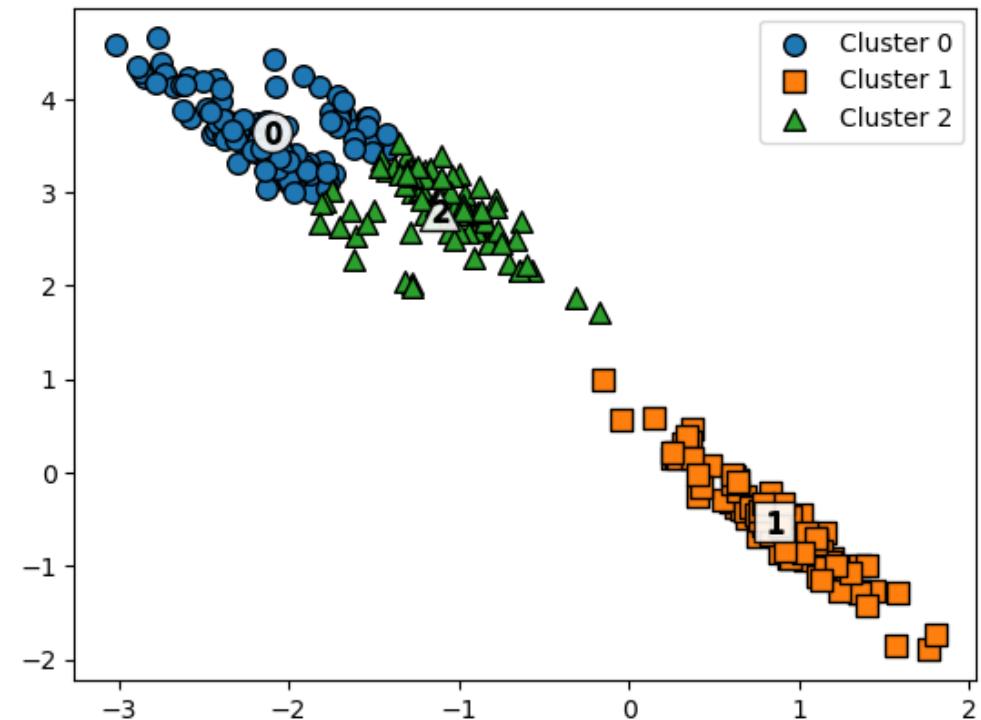
타원형 군집화 데이터 생성



# GMM과 K-Means 비교

```
# 3개의 Cluster 기반 Kmeans 를 X_aniso 데이터 셋에 적용  
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)
```

→ Kmeans는 원형 영역 위치로 개별 군집화를 진행하면서  
원하는 방향으로 군집화 되지 않음

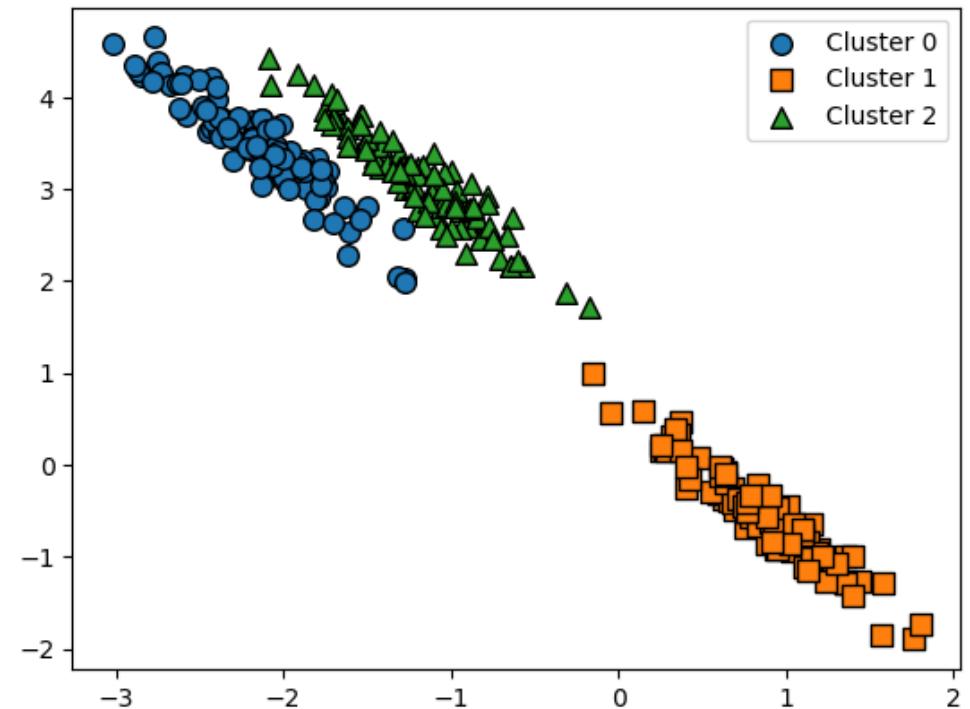


# GMM과 K-Means 비교

```
# 3개의 n_components기반 GMM을 X_aniso 데이터 셋에 적용
gmm = GaussianMixture(n_components=3, random_state=0)
gmm_label = gmm.fit(X_aniso).predict(X_aniso)
clusterDF['gmm_label'] = gmm_label

# GaussianMixture는 cluster_centers_ 속성이 있으므로 iscenter를 False로 설정.
visualize_cluster_plot(gmm, clusterDF, 'gmm_label', iscenter=False)
```

→ GMM는 각 데이터의 정규분포로 군집화 파악. 데이터가  
분포된 방향에 따라 정확하게 군집화



05.

DBSCAN

# 주요 파라미터와 데이터 포인트

- 주요 파라미터

입실론 주변 영역 (epsilon) : 개별 데이터를 중심으로 입실론 반경을 가지는 원형의 영역

최소 데이터 개수 (min points) : 개별 데이터의 입실론 주변 영역에 포함되는 타 데이터의 개수

- 데이터 포인트

핵심 포인트(Core Point): 주변 영역 내에 최소 데이터 개수 이상의 타 데이터를 가지고 있을 경우

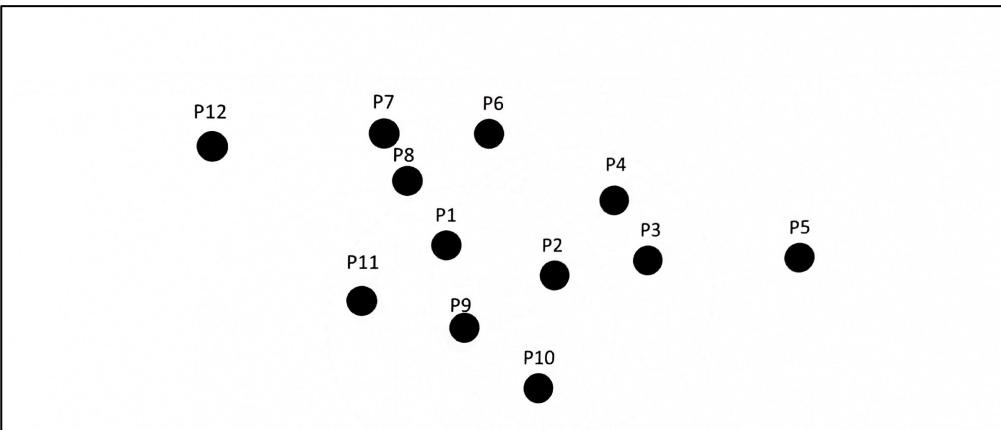
이웃 포인트 (Neighbor Point): 주변 영역 내에 위치한 타 데이터

경계 포인트 (Border Point): 주변 영역 내에 최소 데이터 개수 이상의 이웃 포인트를 가지고 있지 않지만 핵심 포인트를 이웃 포인트로 가지고 있는 데이터

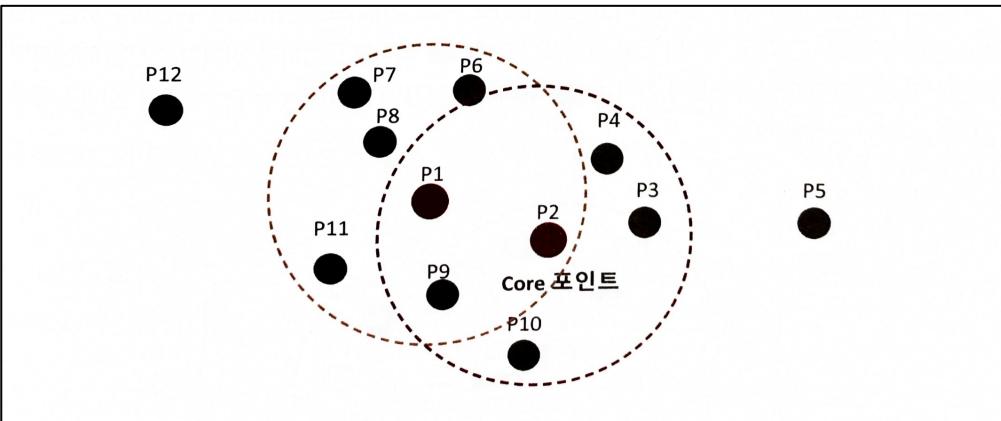
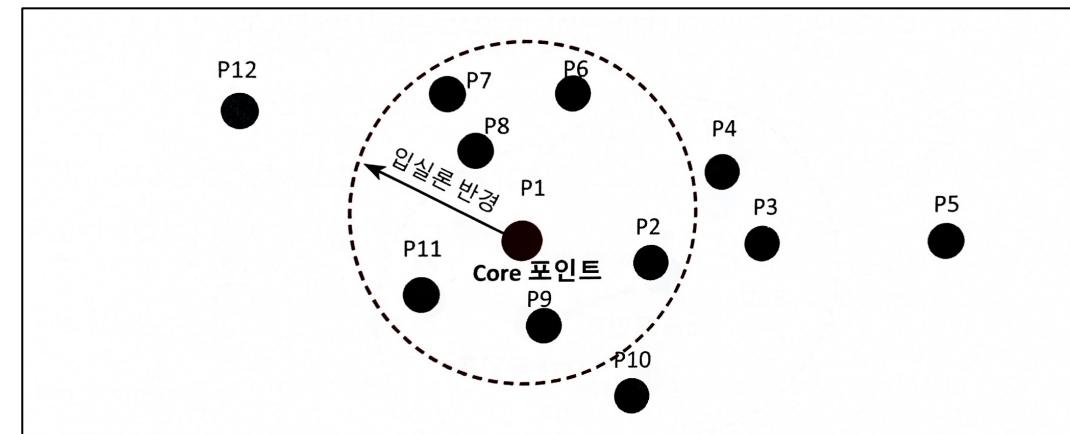
잡음 포인트 (Noise Point): 최소 데이터 개수 이상의 이웃 포인트를 가지고 있지 않으며, 핵심 포인트도 이웃 포인트로 가지고 있지 않는 데이터

# DBSCAN

1. 특정 입실론 반경 내에 포함될 최소 데이터 개수를 6개로 가정



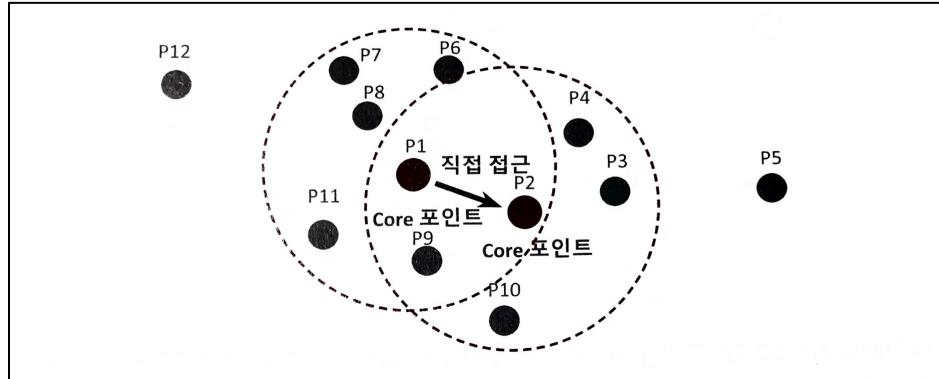
2. P1 데이터를 기준으로 입실론 반경 내에 포함된 데이터가 7개이므로 P1 데이터는 핵심 포인트



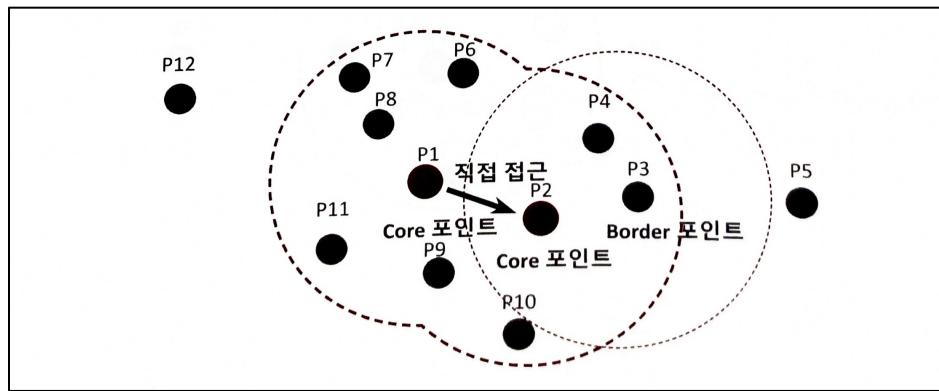
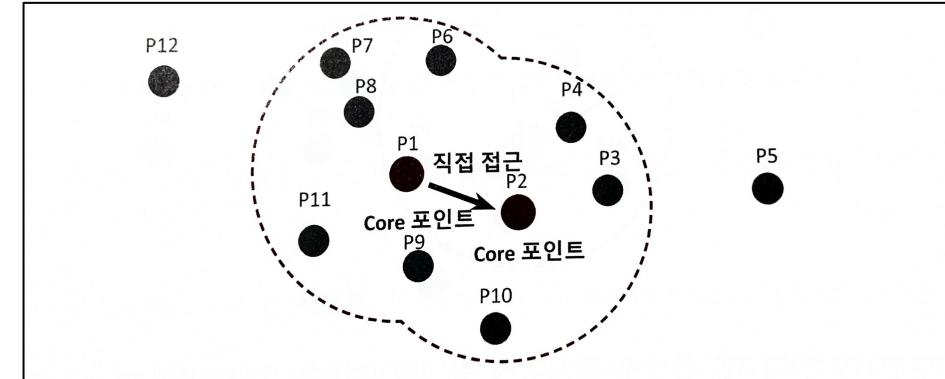
3. P2 데이터 포인트도 역시 반경 내에 6개의 데이터를 가지고 있으므로 핵심 포인트

# DBSCAN

4. P1 이웃 데이터 포인트 P2도 핵심포인트이므로 P1에서 P2로 연결 가능



5. 특정 핵심 포인트에서 다른 핵심포인트를 서로 연결하면서 영역을 확장 → 군집 형성



6. 경계 포인트는 군집의 외각 결정, 핵심 포인트가 이웃 데이터로 가지지 않는 포인트를 잡음 포인트라 함

# 붓꽃 데이터에 DBSCAN 적용하기

1. (eps = 0.6, min\_samples=8)로 군집화

```
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

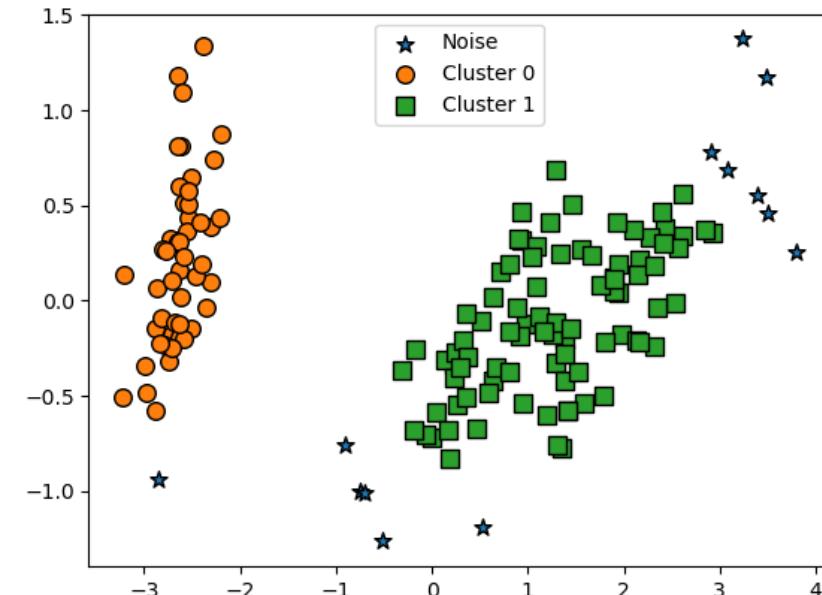
iris_result = irisDF.groupby(['target'])['dbscan_cluster'].value_counts()
print(iris_result)

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

2. PCA 이용해 2개의 피처로 압축 변환 후 시각화

```
from sklearn.decomposition import PCA
# 2차원으로 시각화하기 위해 PCA n_components=2로 피처 데이터 세트 변환
pca = PCA(n_components=2, random_state=0)
pca_transformed = pca.fit_transform(iris.data)
# visualize_cluster_2d() 함수는 ftr1, ftr2 컬럼을 좌표에 표현하므로 PCA 변환값을 해당 컬럼으로 생성
irisDF['ftr1'] = pca_transformed[:,0]
irisDF['ftr2'] = pca_transformed[:,1]

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



# 붓꽃 데이터에 DBSCAN 적용하기

3. ( $\text{eps} = 0.8$ ,  $\text{min\_samples}=8$ )로 군집화

```
from sklearn.cluster import DBSCAN

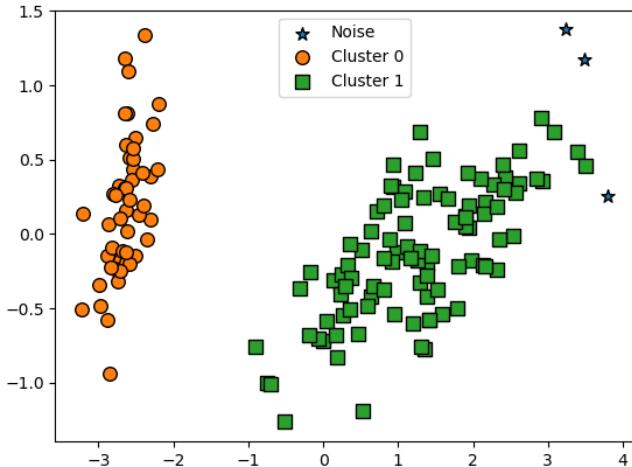
dbscan = DBSCAN(eps=0.8, min_samples=8, metric='euclidean')
dbscan_labels = dbscan.fit_predict(iris.data)

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)

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



4. ( $\text{eps} = 0.6$ ,  $\text{min\_samples}=16$ )로 군집화

```
dbscan = DBSCAN(eps=0.6, min_samples=16, metric='euclidean')
dbscan_labels = dbscan.fit_predict(iris.data)

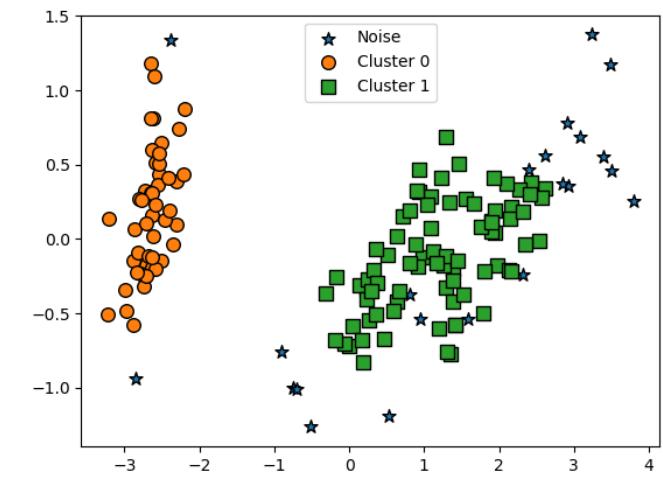
irisDF['dbscan_cluster'] = dbscan_labels
irisDF['target'] = iris.target

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

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

target	dbscan_cluster	count
0	0	48
	-1	2
1	1	44
	-1	6
2	1	36
	-1	14

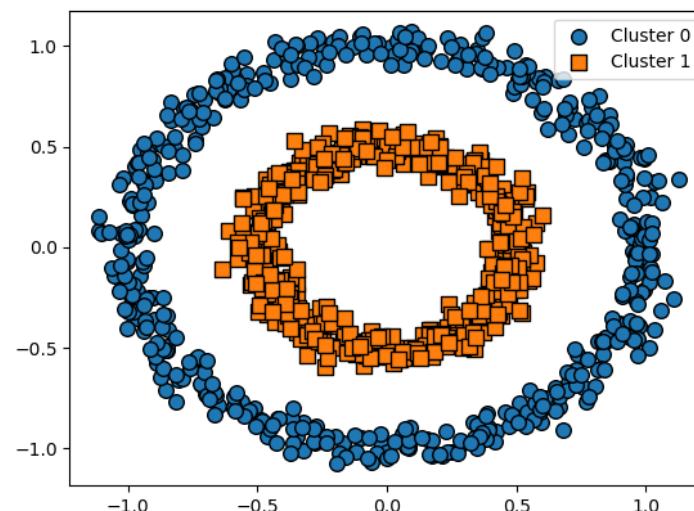
Name: count, dtype: int64



# DBSCAN의 적용 - make\_circles()

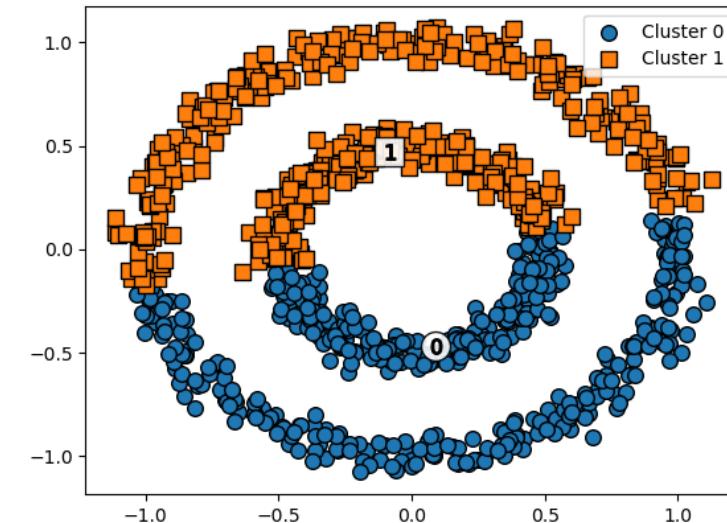
## 1. make\_circles() 함수

```
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)
```



## 2. K-means

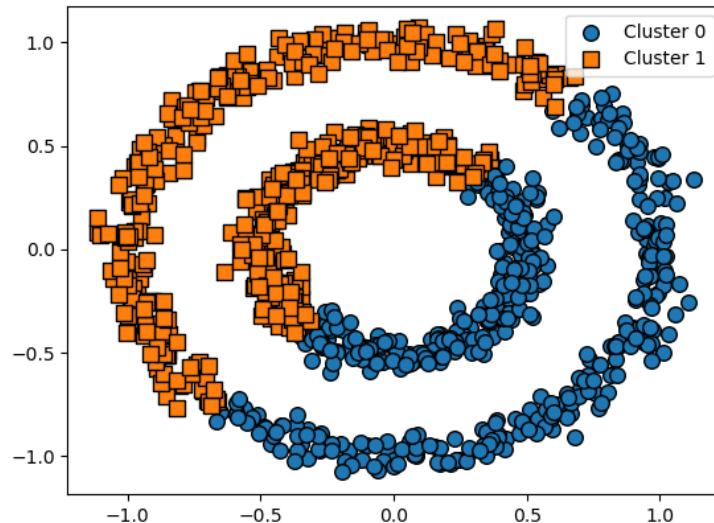
```
# KMeans로 make_circles( ) 데이터 셋을 클러스터링 수행.  
from sklearn.cluster import KMeans  
  
kmeans = KMeans(n_clusters=2, max_iter=1000, random_state=0)  
kmeans_labels = kmeans.fit_predict(X)  
clusterDF['kmeans_cluster'] = kmeans_labels  
  
visualize_cluster_plot(kmeans, clusterDF, 'kmeans_cluster', iscenter=True)
```



# DBSCAN의 적용 - make\_circles()

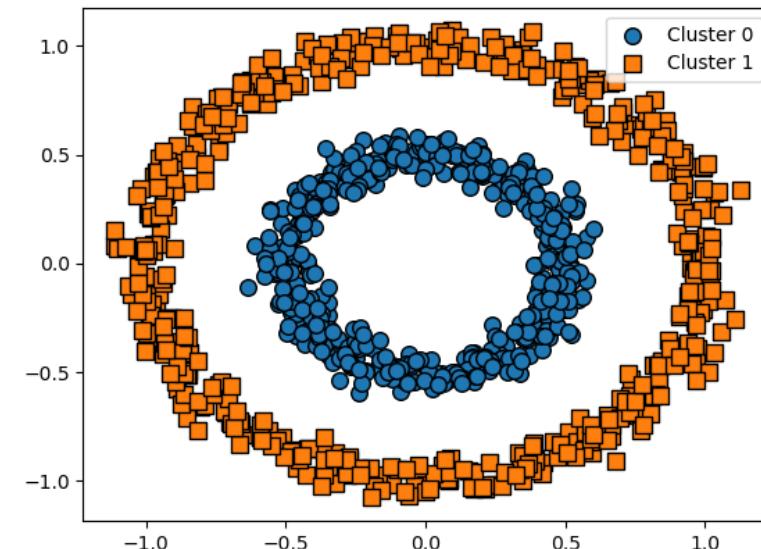
## 3. GMM

```
# 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)
```



## 4. DBSCAN

```
# 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)
```



수고하셨습니다