07 군집화

01 K-평균 알고리즘 이해

K-평균은 군집화에서 가장 일반적으로 사용되는 알고리즘이다.

군집 중심점이라는 특정한 임의의 지점을 선택해 해당 중심에 가장 가까운 포인트들을 선택하는 군집화 기법이다.

1. 먼저 군집화의 기준이 되는 중심을 구성하려는 군집화 개수만큼 임의의 위치에 가져다 놓는다.
2. 각 데이터는 가장 가까운 곳에 위치한 중심점에 소속된다.
3. 군집 중심점은 소속된 데이터의 평균 중심으로 이동한다.
4. 각 데이터는 기존에 속한 중심점보다 더 가까운 중심점이 있다면 해당 중심점으로 다시 소속을 변경한다.
5. 다시 중심을 소속된 데이터의 평균 중심으로 이동한다.
6. 중심점을 이동했는데 데이터의 중심점 소속 변경이 없으면 군집화를 종료한다. 그렇지 않으면 4번 과정을 거쳐서 소속을 변경하고 이 과정을 반복한다.

K-평균의 장점

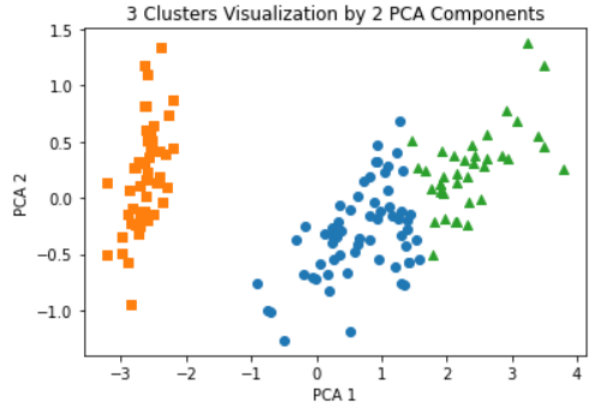
- 일반적인 군집화에서 가장 많이 활용되는 알고리즘이다.
- 알고리즘이 쉽고 간결하다

K-평균의 단점

- 거리 기반 알고리즘으로 속성의 개수가 많을 경우 군집화 정확도가 떨어진다.
- 반복을 수행하는데, 반복 횟수가 많을 경우 수행 시간이 매우 느려진다.
- 몇 개의 군집을 선택해야 할지 가이드하기가 어렵다.

K- 평균을 이용한 붓꽃 데이터 세트 군집화

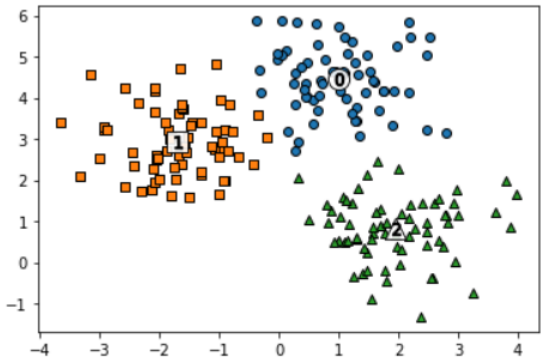
```
# cluster 값이 0, 1, 2 인 경우마다 별도의 Index로 추출
marker0_ind = irisDF[irisDF['cluster']==0].index
marker1_ind = irisDF[irisDF['cluster']==1].index
marker2_ind = irisDF[irisDF['cluster']==2].index

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

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



군집화 알고리즘 테스트를 위한 데이터 생성

`make_blob()` : 개별 군집의 중심점과 표준 편차 제어 기능을 가지고 여러 개의 클래스에 해당하는 데이터 세트를 만든 후 하나의 클래스에 여러 개의 군집이 분포될 수 있게 데이터를 생성하는 것

```
# KMeans 객체를 이용하여 X 데이터를 K-Means 클러스터링 수행
kmeans = KMeans(n_clusters=3, init='k-means++', max_iter=200, random_state=0)
cluster_labels = kmeans.fit_predict(X)
clusterDF['kmeans_label']  = cluster_labels

#cluster_centers_ 는 개별 클러스터의 중심 위치 좌표 시각화를 위해 추출
centers = kmeans.cluster_centers_
unique_labels = np.unique(cluster_labels)
markers=['o', 's', '^', 'P','D','H','x']

# 군집된 label 유형별로 iteration 하면서 marker 별로 scatter plot 수행.
for label in unique_labels:
    label_cluster = clusterDF[clusterDF['kmeans_label']==label]
    center_x_y = centers[label]
    plt.scatter(x=label_cluster['ftr1'], y=label_cluster['ftr2'], edgecolor='k',
                marker=markers[label] )

    # 군집별 중심 위치 좌표 시각화
    plt.scatter(x=center_x_y[0], y=center_x_y[1], s=200, 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)

plt.show()
```




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

```
target  kmeans_label
0       0              66
        1               1
1       2              67
2       1              65
        2               1
Name: kmeans_label, dtype: int64
```

02 군집 평가(Cluster Evaluation)

군집화는 데이터 내의 숨어 있는 별도의 그룹을 찾아서 의미를 부여하거나 동일한 분류 값에 속하더라도 그 안에서 더 세분화된 군집화를 추구하거나 서로 다른 분류 값의 데이터도 더 넓은 군집화 레벨화 등의 영역을 가지고 있다.

실루엣 분석의 개요

실루엣 분석은 각 군집 간의 거리가 얼마나 효율적으로 분리돼 있는지를 나타낸다.

효율적으로 잘 분리가 되었다는 것

→ 다른 군집과의 거리가 떨어져 있다.

→ 동일 군집끼리의 데이터는 서로 가깝게 잘 뭉쳐 있다.

실루엣 분석은 실루엣 계수를 기반으로 한다. **실루엣 계수는 개별 데이터가 가지는 군집화 지표**이다.

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

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

실루엣 계수는 -1에서 1 사이의 값을 가지며, 1로 가까워질 수록 근처의 군집과 더 멀리 떨어져 있다는 것이고 0에 가까울수록 근처의 군집과 가까워진다는 것이다. - 값은 아예 다른 군집에 데이터 포인트가 할당되었음을 말한다.

붓꽃 데이터 세트를 이용한 군집 평가

```
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(irisDF)

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	0.851573
1	4.9	3.0	1.4	0.2	1	0.817887
2	4.7	3.2	1.3	0.2	1	0.830087

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

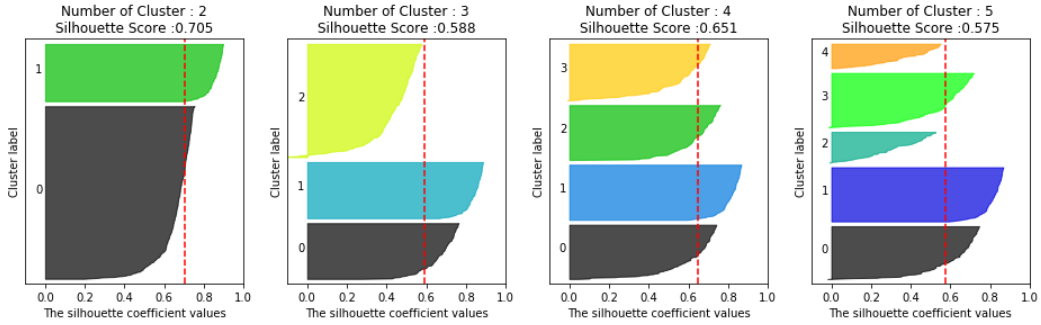
```
cluster
0    0.417182
1    0.797630
2    0.451105
Name: silhouette_coeff, dtype: float64
```

1번 군집은 실루엣 계수 평균 값이 약 0.79인데 반해, 0번은 약 0.41, 2번은 0.45로 상대적으로 평균값이 1번에 비해 낮다.

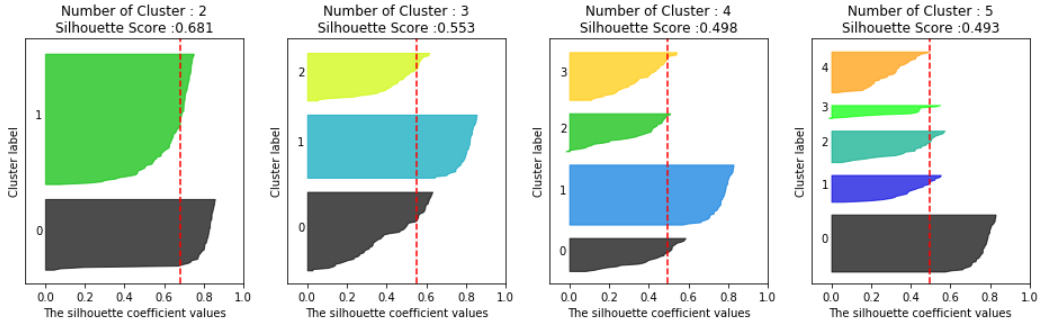
군집별 평균 실루엣 계수의 시각화를 통한 군집 개수 최적화 방법

여러 개의 군집 개수가 주어졌을 때, 평균 실루엣 계수로 군집 개수를 최적화하는 방법을 알아보자.

```
# make_blobs 을 통해 clustering 을 위한 4개의 클러스터 중심의 500개 2차원 데이터 셋 생성
from sklearn.datasets import make_blobs
X, y = make_blobs(n_samples=500, n_features=2, centers=4, cluster_std=1, ₩
                  center_box=(-10.0, 10.0), shuffle=True, random_state=1)

# cluster 개수를 2개, 3개, 4개, 5개 일때의 클러스터별 실루엣 계수 평균값을 시각화
visualize_silhouette([ 2, 3, 4, 5], X)
```




```
from sklearn.datasets import load_iris

iris=load_iris()
visualize_silhouette([ 2, 3, 4,5 ], iris.data)
```



`make_blobs()` 함수를 통해 4개 군집 중심의 500개 2차원 데이터 세트를 만들고 이를 K-평균으로 군집화할 때, 4개의 군집일 때 가장 최적이 됨을 알 수 있다.

붓꽃 데이터를 K-평균으로 군집화할 경우에는 군집 개수를 2개로 하는 것이 가장 좋아보인다.

03 평균 이동

평균 이동(Mean Shift)의 개요

평균이동은 중심을 군집의 중심으로 지속적으로 움직이면서 군집화를 수행한다.

→ 중심을 데이터가 모여 있는 밀도가 가장 높은 곳으로 이동시킨다.

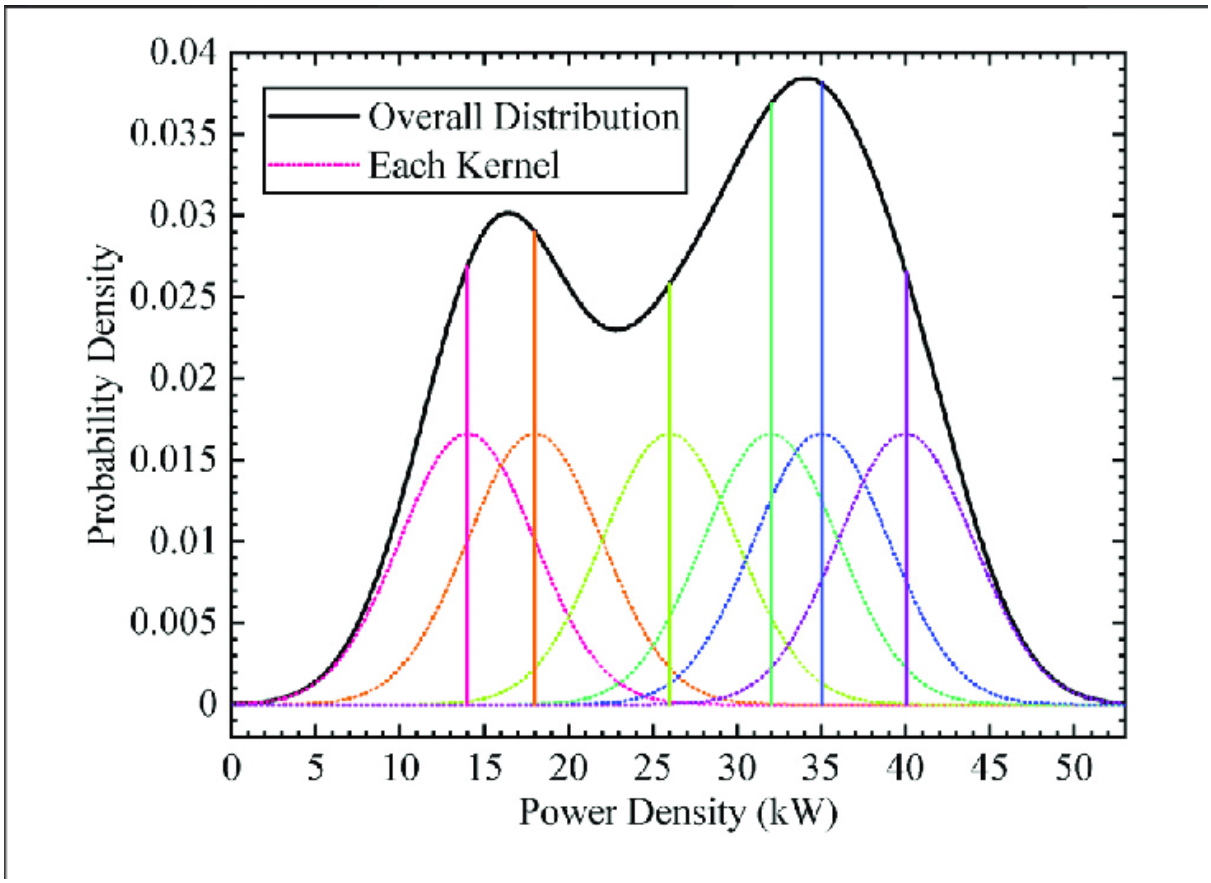
평균 이동 군집화는 데이터의 분포도를 이용해 군집 중심점을 찾는다.

군집 중심점은 데이터 포인터가 모여있는 곳이며, 이를 위해 **확률밀도함수를 이용**한다.

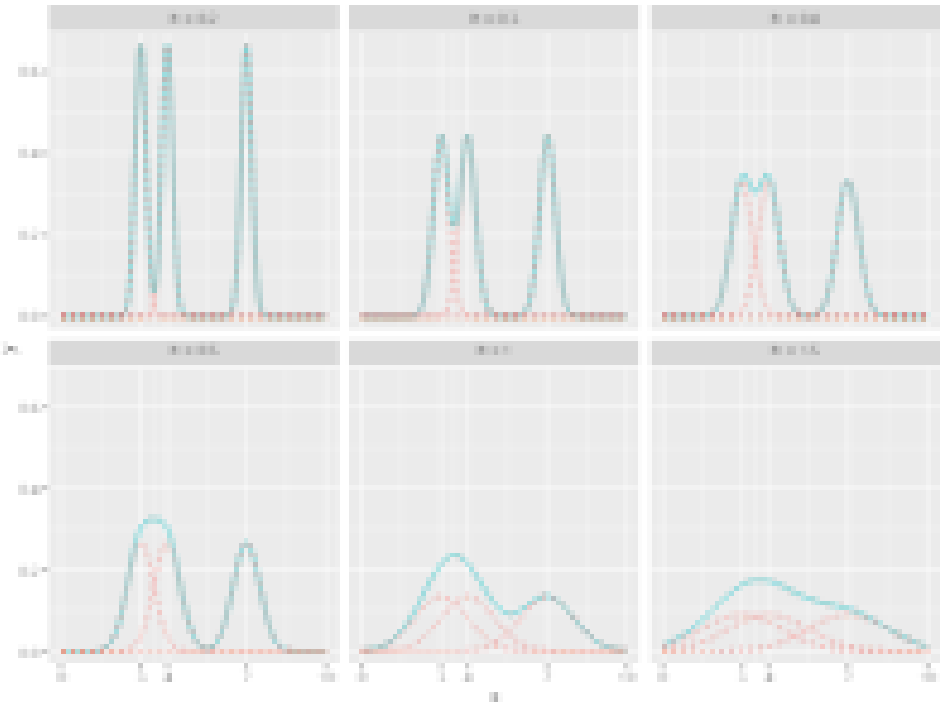
확률밀도함수가 피크인 곳을 군집중심점으로 선정하며 일반적으로 주어진 모델의 확률 밀도 함수를 찾기 위해서 KDE를 이용한다.

KDE를 활용한 군집 중심점 찾기

1. 개별 데이터의 특정 반경 내에 주변 데이터를 포함한 데이터 분포도를 **KDE 기반의 Mean Shift 알고리즘으로 계산**한다.
2. KDE로 계산된 **데이터 분포도가 높은 방향으로 데이터 이동**
3. 모든 데이터를 1~2까지 수행하면서 데이터를 이동. 개별 데이터들이 군집중심점으로 모인다.
4. 지정된 반복 횟수만큼 전체 데이터에 대해서 KDE 기반으로 **데이터를 이동시키면서 군집화 수행**
5. 개별 데이터들이 모인 중심점을 군집 중심점으로 설정



가우시안 커널 함수 적용 후 합산한 KDE 결과



Bandwidth에 따른 KDE 기반의 평균이동 군집화

`estimate_bandwidth()` 함수를 통해 최적의 대역폭 계산 후 나타낸 군집화 결과를 살펴보자.

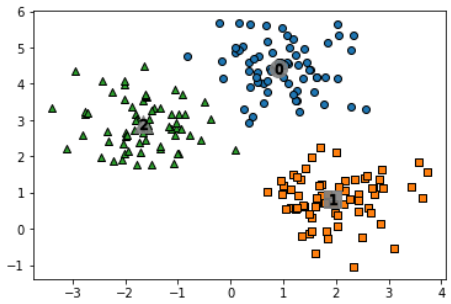
```
from sklearn.cluster import estimate_bandwidth

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

```
bandwidth 값: 1.816
```

```
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))
```

```
cluster labels 유형: [0 1 2]
```

```
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]
    # 군집별로 다른 마커로 산점도 적용
    plt.scatter(x=label_cluster['ftr1'], y=label_cluster['ftr2'], edgecolor='k', marker=markers[label] )

    # 군집별 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()
```



최적의 Bandwidth는 1.816, 그에 따른 cluster label은 3개의 군집으로 잘 구성되었음을 알 수 있다.

`cluster_centers_` 속성으로 군집 중심 좌표를 표시하고 위와 같은 결과가 나옴을 알 수 있다.

평균 이동의 장점

- 데이터 세트의 형태를 특정 형태로 가정한다든가, 특정 분포도 기반의 모델로 가정하지 않기 때문에, 좀 더 유연한 군집화가 가능하다.
- 이상치의 영향력도 크지 않으며, 미리 군집의 개수를 정할 필요도 없다.

평균 이동의 단점

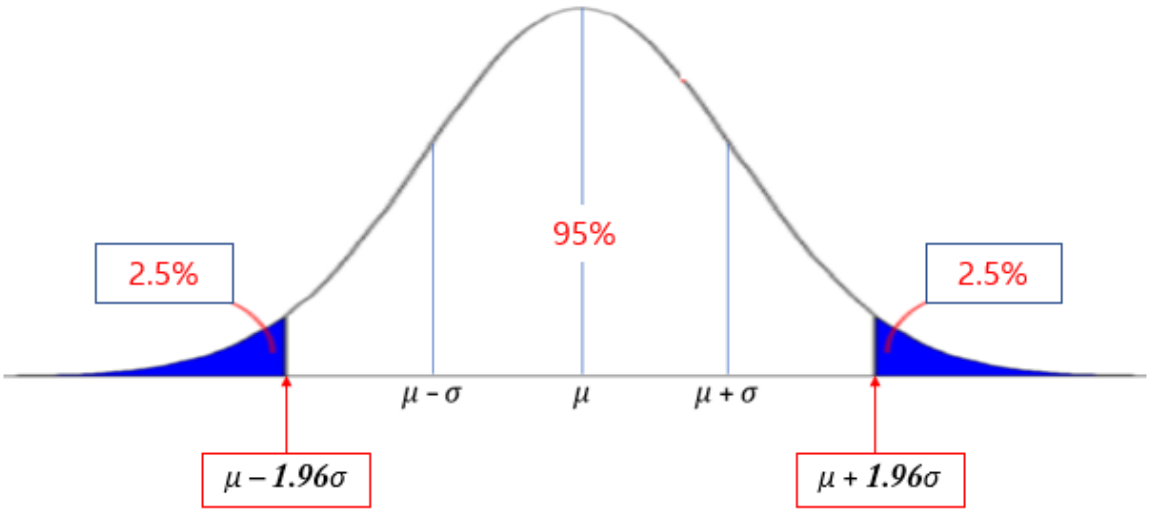
- 알고리즘의 수행 시간이 오래 걸리고, 무엇보다도 Band-width의 크기에 따른 군집화 영향도가 매우 크다.

  → 따라서 CV 영역에서 더 뛰어난 역할을 수행할 수 있다.

04 GMM(Gaussian Mixture Model)

GMM 소개

GMM 군집화는 군집화를 적용하고자 하는 데이터가 여러 개의 가우시안 분포를 가진 데이터 집합들이 섞여서 생성된 것이라는 가정하에 군집화를 수행하는 방식이다.



전체 데이터 세트는 서로 다른 정규 분포 형태를 가진 여러 가지 확률 분포 곡선으로 구성될 수 있으며, 이러한 서로 다른 정규 분포에 기반해 군집화를 수행하는 것이 **모수 추정**(GMM 군집화 방식)이다.

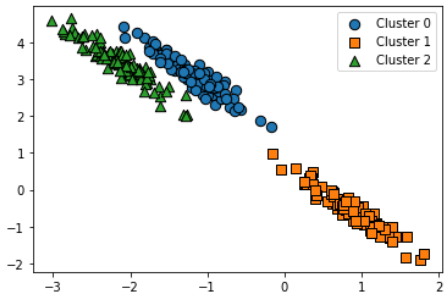
모수 추정

- 개별 정규 분포의 평균과 분산
- 각 데이터가 어떤 정규 분포에 해당되는지의 확률

GMM과 K-평균의 비교

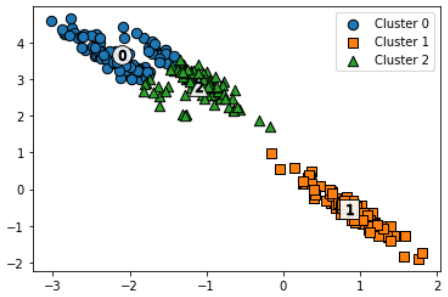
k-means는 데이터의 분포가 원형일 때 효과적으로 군집화를 수행할 수 있다. 만일 `mark_blobs`로 군집을 만들 때 `cluster_std`를 작게 설정한다면, 데이터의 분포는 원형의 모습을 가지게 된다. 하지만 k-means는 데이터가 길쭉한 타원형으로 늘어선 경우에 군집화를 잘 수행하지 못한다.

```
from sklearn.datasets import make_blobs

# make_blobs() 로 300개의 데이터 셋, 3개의 cluster 셋, cluster_std=0.5 을 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 marker 로 표시하여 시각화 함.
visualize_cluster_plot(None, clusterDF, 'target', iscenter=False)
```



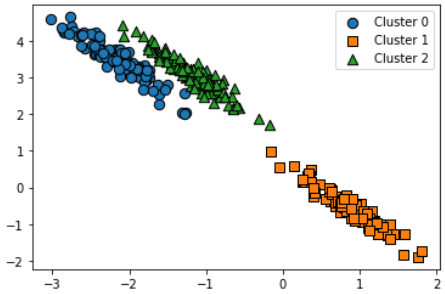
위의 데이터 세트는 KMeans의 군집화 정확성이 떨어지게 된다. KMeans가 위의 데이터 세트를 어떻게 군집화하는지 확인해보자.

```
# 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)
```



위의 데이터 상으로 **군집화 분류가 제대로 되지 않았음**이 보인다.

```
# 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 없으므로 iscenter를 False로 설정.
visualize_cluster_plot(gmm, clusterDF, 'gmm_label',iscenter=False)
```



GMM분포를 이용한 군집화를 통해서 분류가 잘 된 모습을 볼 수 있다.

```
### 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 군집 1번만 정확히 매핑되었고, 나머지 군집의 경우 target 값과 어긋나는 경우가 발생하고 있다. 하지만 GMM의경우는 군집이 target값과 잘 매핑되어있다.

→ KMeans 보다 유연하게 다양한 데이터 세트에 잘 적용될 수 있다는 장점이 있다. 하지만, 군집화를 위한 수행 시간이 오래 걸린다는 단점이 있다.

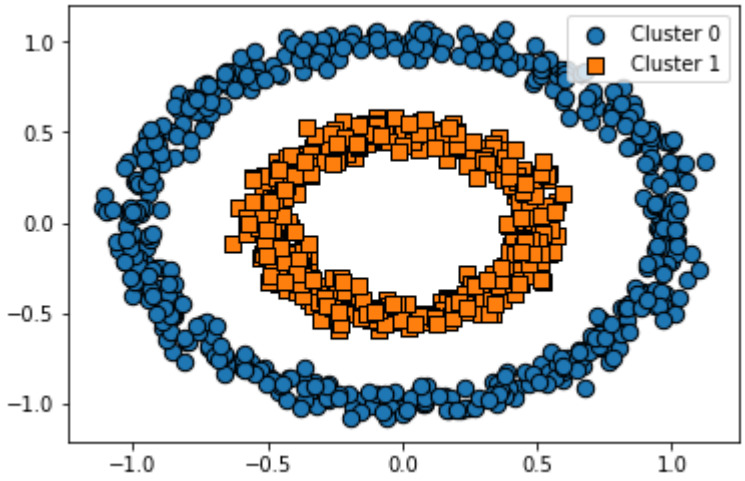
05 DBSCAN

DBSCAN 개요

간단하고 직관적인 알고리즘으로 돼있음에도 데이터의 분포가 기하학적으로 복잡한 데이터 세트에도 효과적인 군집화가 가능하다.

다음과 같이 **내부의 원 모양과 외부의 원 모양 형태의 분포를 가진 데이터 세트를 군집화한다**고 가정할 때, 앞에서 소개한 k 평균, 평균 이동, GMM으로는 효과적인 군집화를 수행하기 어렵다.

DBSCAN은 **특정 공간 내에 데이터 밀도 차이를 기반 알고리즘**으로 하고 있어서 복잡한 기하학적 분포도를 가진 데이터 세트에 대해서도 군집화를 잘 수행한다.

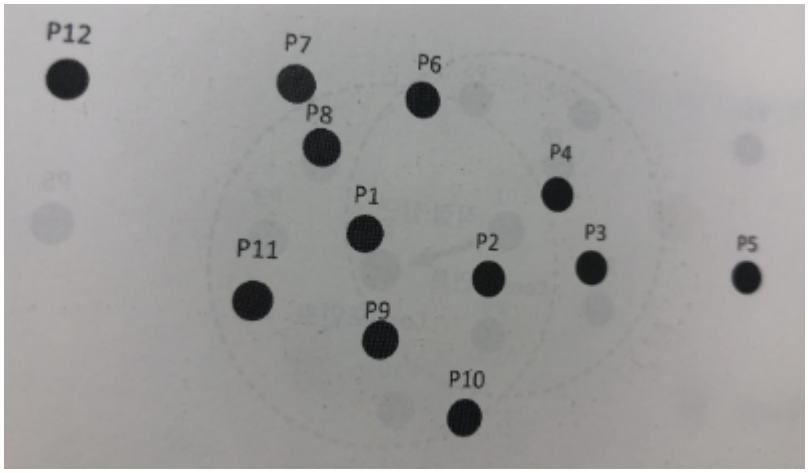


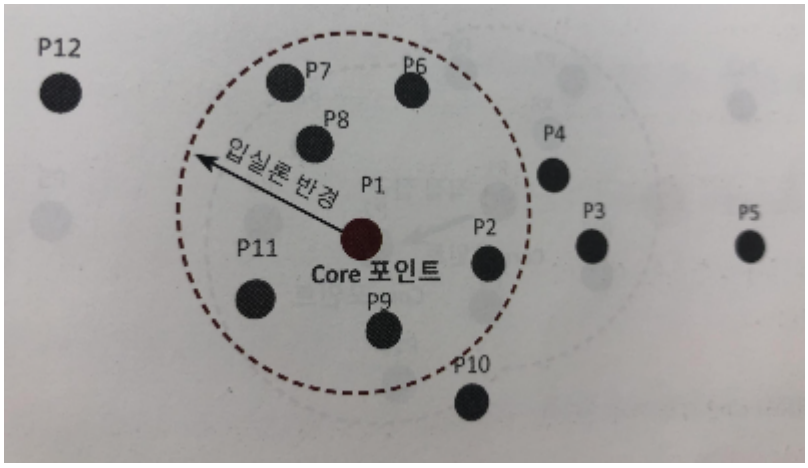
DBSCAN을 구성하는 가장 중요한 두 가지 파라미터는 입실론(epsilon)으로 표기하는 주변 영역과 이 입실론 주변 영역에 포함되는 최소 데이터 개수이다.

데이터 세트 내의 데이터 포인트에 대해서 입실론 내 최소 데이터의 개수에 따라 다르게 정의내린다.

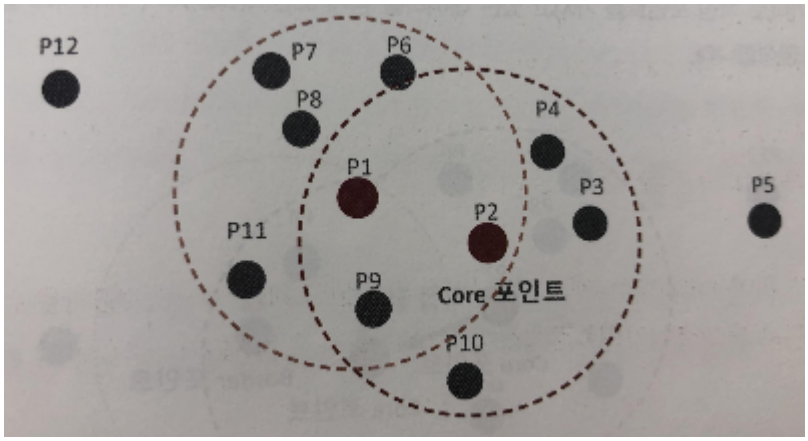
1. 핵심 포인트(core point) : 입신론 주변 영역 내에 최소 데이터 개수 이상의 데이터를 가지고 있을 경우 해당 데이터를 핵심 포인트라고 일컫는다.
2. 이웃 포인트(Neighbor point) : 주변 영역 내에 위치한 타 데이터를 이웃 포인트라고 한다.
3. 경계 포인트(Border point) : 주변 영역 내에 최소 데이터 개수를 만족시키고 있지 않지만 해당 데이터의 경계 내에 핵심포인트가 있다면 해당 데이터를 경계 포인트라고 칭한다.
4. 잡음 포인트(Noise point) : 해당 데이터의 영역 내에 최소 데이터 개수도 충족시키지 못하고 핵심 포인트도 포함하지 못할 경우 해당 데이터를 잡음 데이터로 칭한다.

다음은 DBSCAN 알고리즘의 원리를 그림으로 나타낸 것이다.

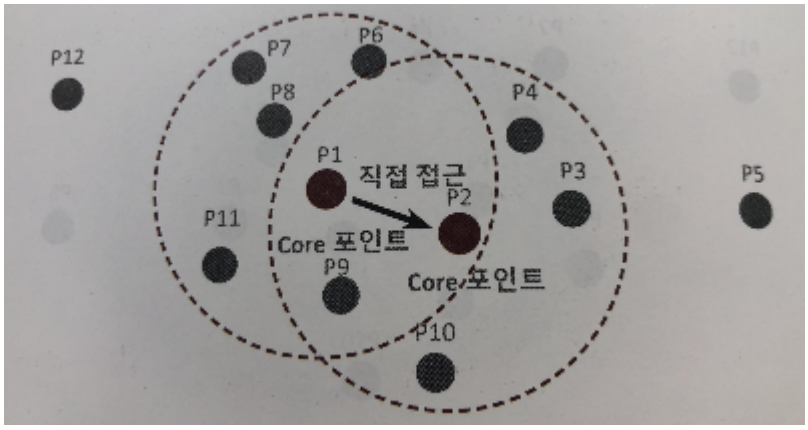




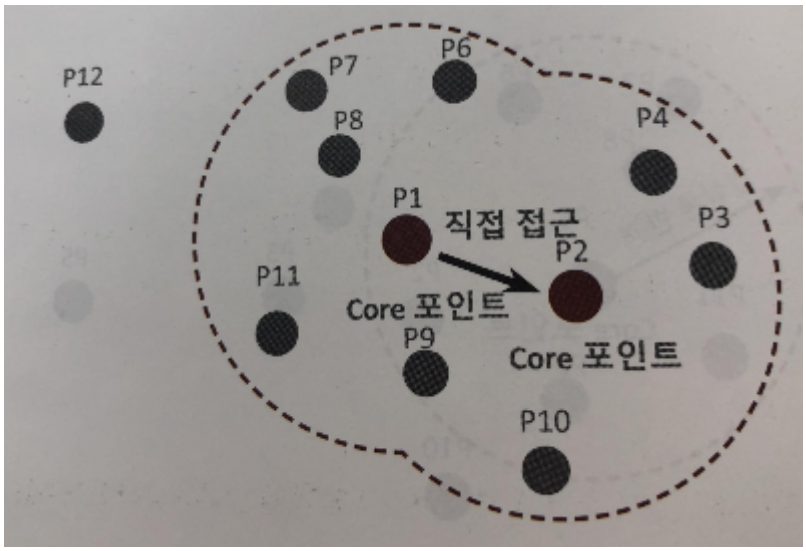
1. 다음 그림과 같이 P1과 P12까지 12개의 데이터 세트에 대해서 DBSCAN 군집화를 적용하면서 특정 입실론 반경 내에 포함될 **최소 데이터 세트를 6개**로 가정해보자.
2. P1 데이터를 기준으로 입실론 반경 내에 포함된 데이터가 7개로 최소 데이터 5개 이상을 만족하므로 P1 데이터 **핵심포인트** 이다.



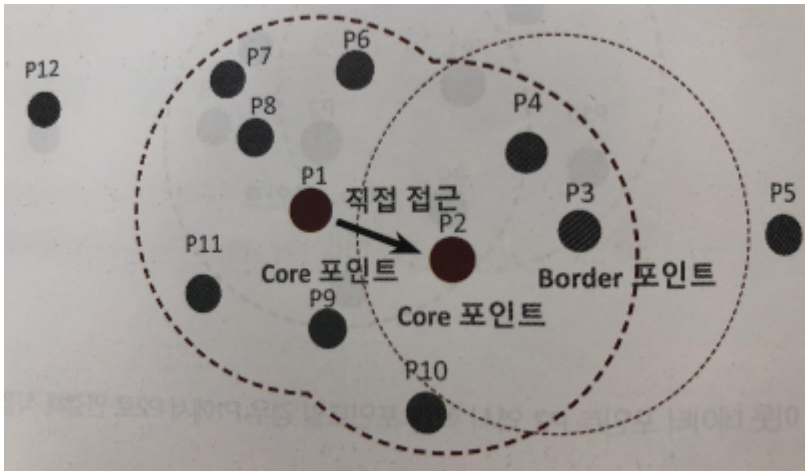
3. 다음으로 P2 데이터 포인트를 살펴보자. P2 반경 내에 6개의 데이터를 가지고 있으므로 핵심 포인트이다.



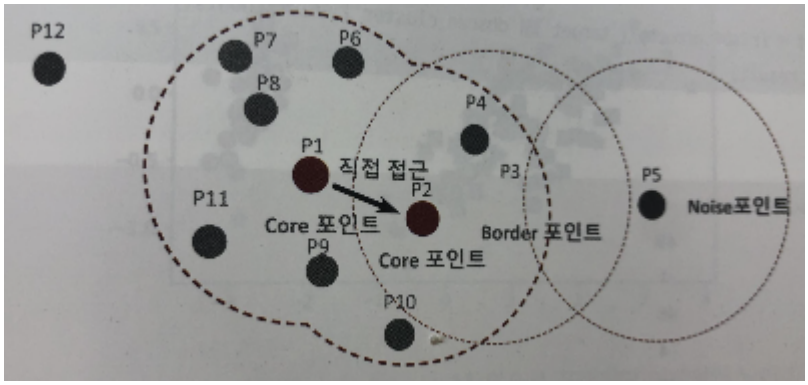
4. **핵심 포인트** P1의 **이웃 데이터 포인트** P2 역시 핵심 포인트일 경우 P1에서 P2로 연결해 직접 접근이 가능하다.



5. 특정 핵심 포인트에서 직접 접근이 가능한 다른 핵심 포인트를 서로 연결하면서 군집화를 구성한다. 이러한 방식으로 점차적으로 군집 영역을 확장해 나가는 것이 DBSCAN 군집화 방식이다.



6. P3 데이터의 경우 반경 내에 포함되는 이웃데이터는 P2, P4로 2개 이므로 군집으로 구분할 수 있는 핵심 포인트가 될 수 없다. 하지만, 이웃 데이터 중에 핵심 포인트인 P2를 가지고 있다. 이처럼 **자신은 핵심포인트가 아니지만 이웃데이터로 핵심 포인터를 가지고 있는 데이터를 경계 포인트**하고 한다.



7. 다음 그림의 P5와 같이 **반경 내에 최소 데이터를 가지고 있지도 않고 핵심 포인트 또한 이웃 데이터로 가지고 있지 않은 데이터**를 잡음 포인트라고 한다.

DBSCAN은 이처럼 **입실론 주변 영역의 최소 데이터 개수를 포함하는 밀도 기준을 충족시키는 데이터인 핵심 포인트를 연결하면서 군집화를 구성하는 방식**이다.

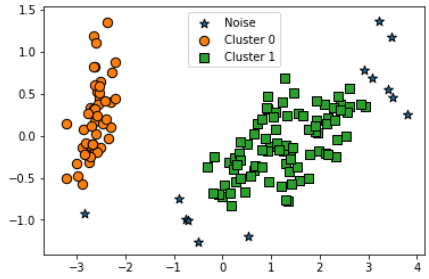
DBSCAN 적용하기 - 붓꽃 데이터 세트

```
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
```

군집 레이블이 -1인 것은 노이즈에 속하는 군집을 의미한다. 따라서 위 붓꽃 데이터 세트는 DBSCAN에서 0과 1 두개의 군집으로 군집화됐다. DBSCAN은 군집의 개수를 알고리즘에 따라 자동을 지정하므로 군집의 개수를 지정하는 것은 무의미하다.

```
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)
```



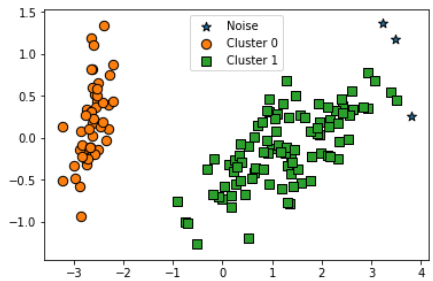
★로 표시된 부분은 모두 노이즈이며, DBSCAN을 적용할 땐 특정 군집 개수로 군집을 강제하지 않는 것이 좋다.

`eps` 값을 크게 하면 주어진 반경 내에서 더 많은 데이터를 포함시켜야 하므로 노이즈 데이터 개수가 작아지며, `min_samples` 를 크게 하면 주어진 반경 내에서 더 많은 데이터를 포함시켜야 하므로 노이즈 데이터 개수가 커지게 된다.

다음으로 `eps`를 0.6에서 0.8로 늘려보자.

```
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: dbscan_cluster, dtype: int64
```

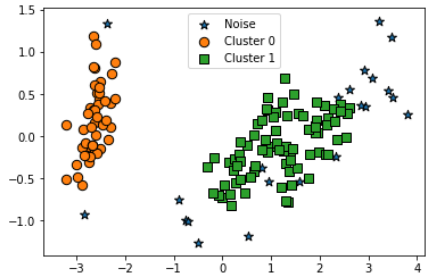


노이즈 군집인 -1이 3개 밖에 없다. 기존 eps가 0.6일 때 노이즈로 분류된 데이터 세트는 eps 반경으로 커지면서 Cluster 1에 소속되었다.

이번에는 `eps` 를 0.6 유지, `min_samples()` 를 16으로 늘려보자.

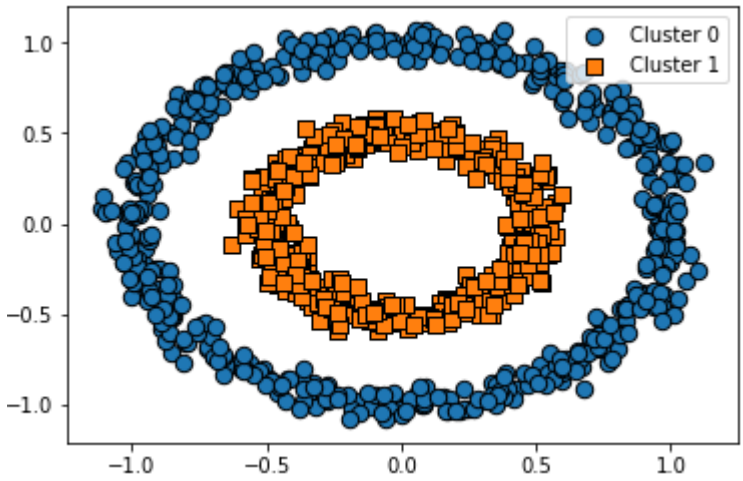
```
dbscan = DBSCAN(eps=0.6, min_samples=16, metric='euclidean')
dbscan_labels = dbscan.fit_predict(iris.data)

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



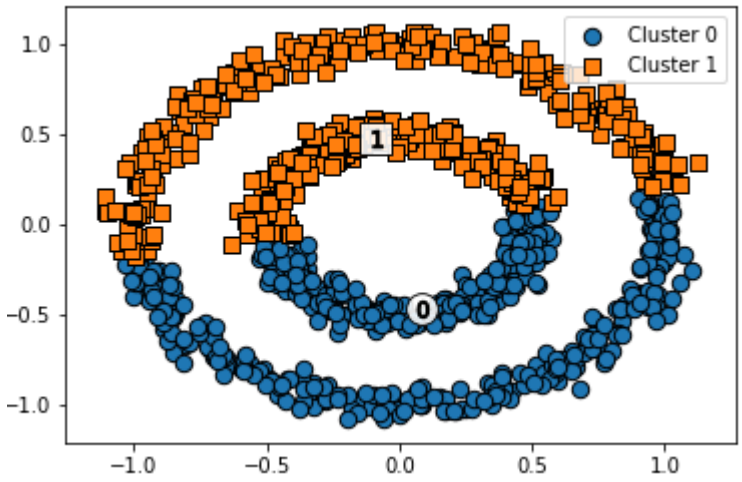
노이즈 데이터가 기존보다 많이 증가함을 알 수 있다.

DBSCAN 적용하기 - make_circles() 데이터 세트



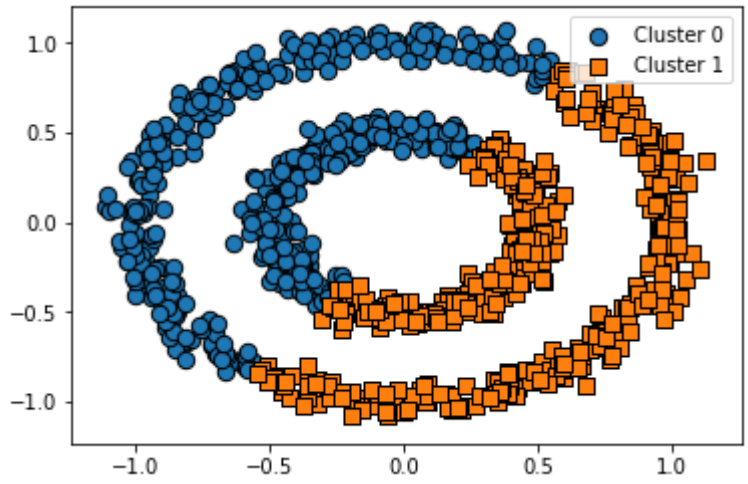
`make_circle()` 를 이용해 내부 원과 외부 원으로 구분되는 데이터 세트를 생성함을 알 수 있다.

먼저 KMeans로 `make_circle()` 데이터 세트를 군집화 해보자.



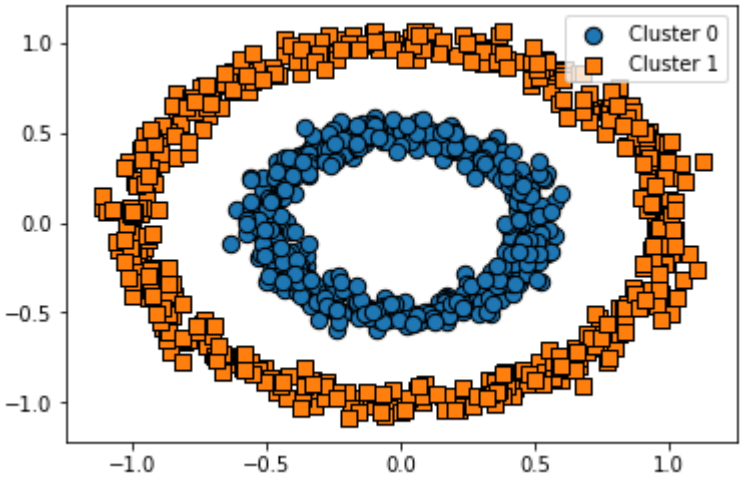
위, 아래 군집 중심을 기반으로 군집화되었다. **거리 기반 군집화로는 위와 같이 데이터가 특정한 형태로 지속해서 이어지는 부분을 찾아내기 어렵다.**

GMM으로 적용해보자.



GMM도 내부와 외부의 원형으로 구성된 더 복잡한 데이터 세트에서는 군집화가 원하는 방향으로 되지 않았다.

DBSCAN으로 적용해보자.



DBSCAN으로 군집화를 적용해 원하는 방향으로 정확히 군집화가 됐음을 알 수 있다.

06 군집화 실습 - 고객 세그먼테이션

고객 세그먼테이션