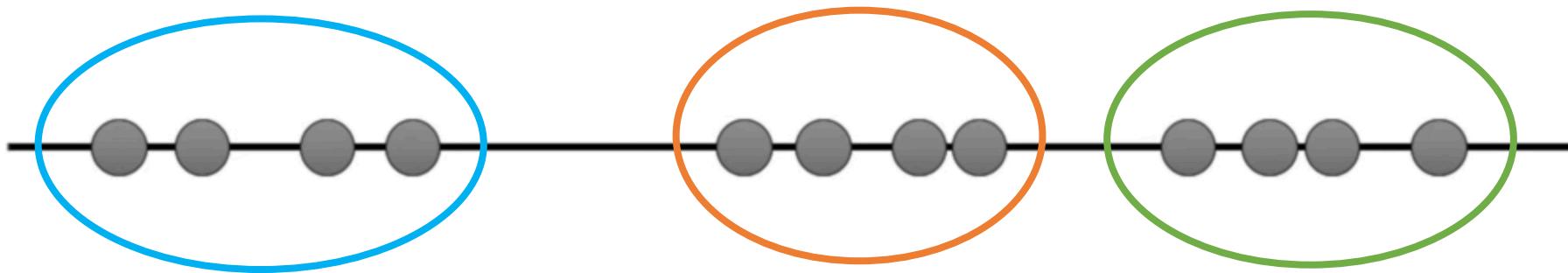


7. 군집화

7.1 ~ 7.3

Kuggle 서민

우리는 군집화(Clustering)를 왜 하는가?



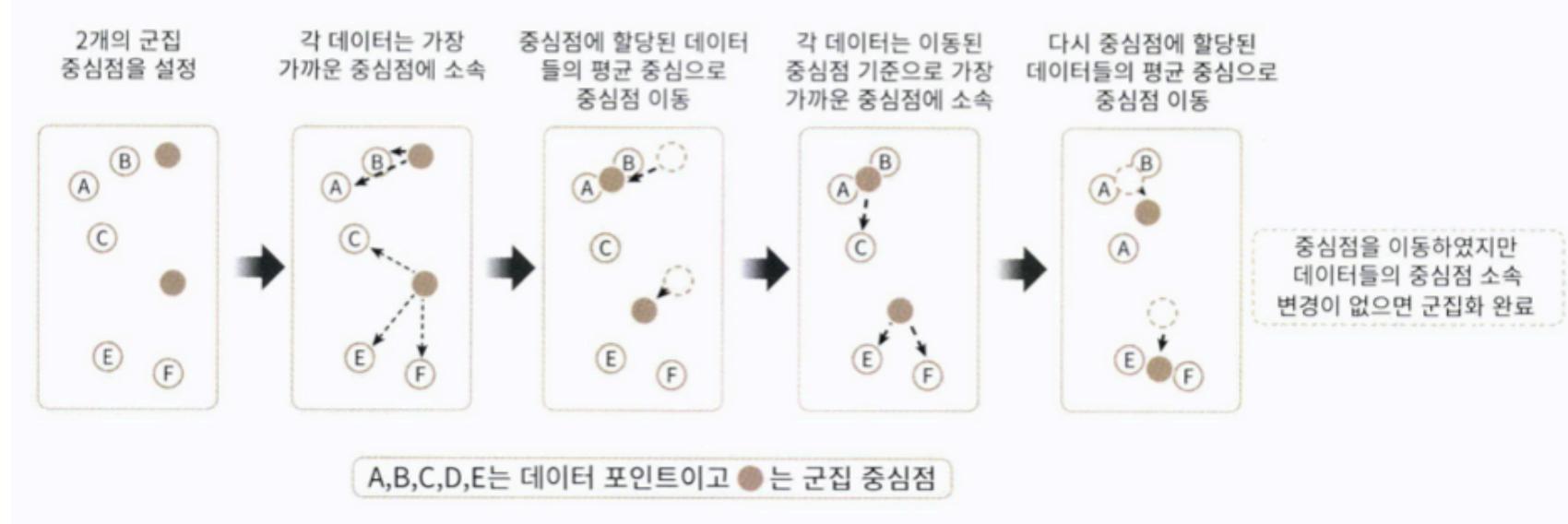
직선의 점들을 분류할 수 있는 방법 = 3가지 = (예를 들어) 서로 다른 암의 종류



데이터의 수가 핵 많을 때 컴퓨터가 이것을 대신해서 비슷한 애들끼리 분류해 주면
비슷한 특성을 가지는 애들끼리 묶이니까 상위 개념을 알 수 있어 당연히 좋겠죠?

7.1_ K- 평균 알고리즘의 이해

- K-평균은 군집화에서 가장 일반적으로 사용되는 알고리즘
- K-평균: 군집 중심점이라는 특정한 임의의 점을 선택해 해당 중심에서 가장 가까운 포인트들을 선택하는 군집화 기법



7.1_ K- 평균 알고리즘의 이해

사이킷런 Kmeans 클래스 소개

가장 중요

```
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=1, algorithm='auto')
```

- n_clusters: 군집화할 개수 = 군집화 중심점의 개수
- init: 초기에 중심점의 좌표를 설정할 방식(주로 K-means++ 쓰임)
- max_iter: 최대 반복 횟수

7.1_ K- 평균 알고리즘의 이해

K-평균을 이용한 붓꽃 데이터 세트 군집화

```
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()
# 더 편리한 데이터 핸들링을 위해 DataFrame으로 변환

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

7.1_ K- 평균 알고리즘의 이해

K-평균을 이용한 붓꽃 데이터 세트 군집화

	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

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

irisDF 데이터를 kmeans 객체변수로 반환

- labels_: 각 데이터 포인트가 속한 군집 중심점 레이블
- cluster_centers_: 각 군집 중심점 좌표 (Shape는 [군집 개수, 피처 개수]) – 이걸로 중심점 좌표 시각화

7.1 K-평균 알고리즘의 이해

K-평균을 이용한 붓꽃 데이터 세트 군집화

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

【Output】

Label 값이 0,1,2 =3개의 군집

- `labels_`: 각 데이터 포인트가 속한 군집 중심점 레이블
 - `cluster centers` : 각 군집 중심점 좌표 (Shape는 [군집 개수, 피처 개수]) – 이걸로 중심점 좌표 시각화

7.1_ K- 평균 알고리즘의 이해

K-평균을 이용한 붓꽃 데이터 세트 군집화

	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

- Label_값을 Cluster칼럼으로 irisDF에 추가
- Group by를 이용하여 target과 cluster값 비교

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

【Output】

	target	cluster
0	1	50
1	0	48
2	2	2
2	0	14
2	2	36

타겟1: 0,2로 군집

타겟2: 0,2로 군집(분산됨)

7.1_ K- 평균 알고리즘의 이해

K-평균을 이용한 붓꽃 데이터 세트 군집화

- 붓꽃데이터 시각화

```
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)
```

	sepal_length	sepal_width	petal_length	petal_width	cluster	pca_x	pca_y
0	5.1	3.5	1.4	0.2	1	-2.684207	0.326607
1	4.9	3.0	1.4	0.2	1	-2.715391	-0.169557
2	4.7	3.2	1.3	0.2	1	-2.889820	-0.137346

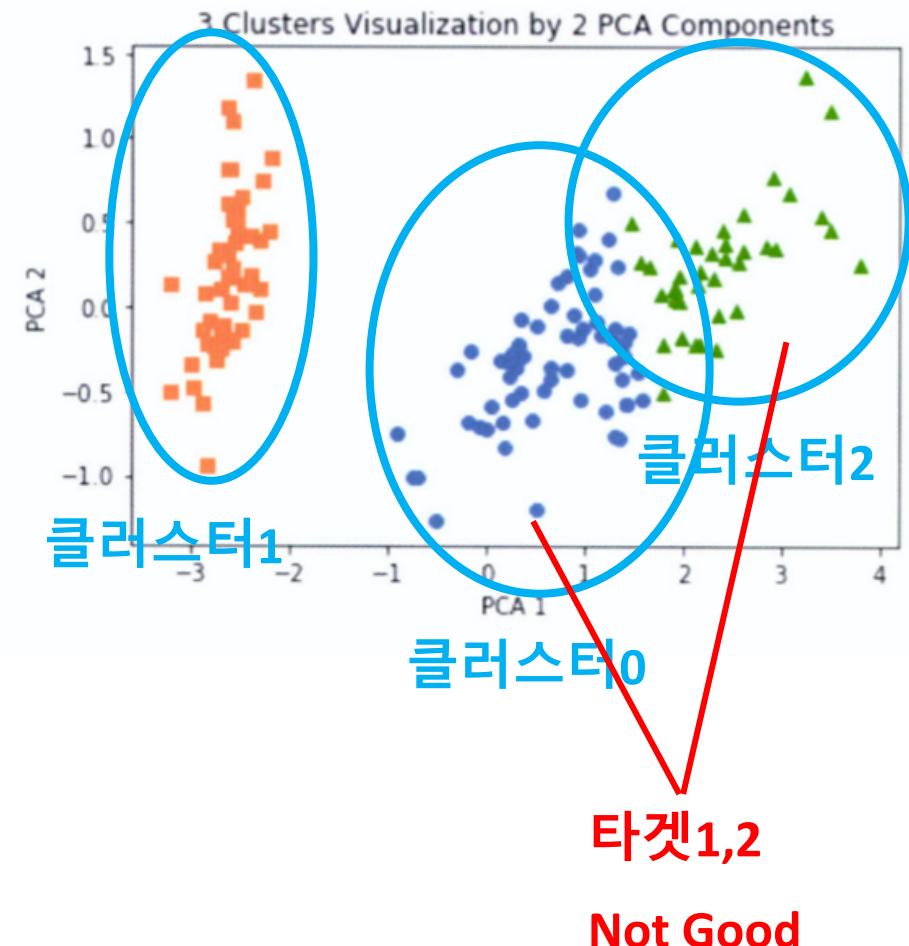
(0,1,2 마커 다른) 마커별로 산점도 다시 그리기

7.1_ K- 평균 알고리즘의 이해

K-평균을 이용한 붓꽃 데이터 세트 군집화

- 마커별 산점도 그리기

```
# 군집 값이 0, 1, 2인 경우마다 별도의 인덱스로 추출  
marker0_ind = irisDF[irisDF['cluster']==0].index  
marker1_ind = irisDF[irisDF['cluster']==1].index  
marker2_ind = irisDF[irisDF['cluster']==2].index  
  
# 군집 값 0, 1, 2에 해당하는 인덱스로 각 군집 레벨의 pca_x, pca_y 값 추출. o, s, ^ 로 마커 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()
```



7.1_ K- 평균 알고리즘의 이해

군집화 알고리즘 테스트를 위한 데이터 생성

- 군집화용 데이터 생성기: make_blobs(), make_classification() API
 - make_blobs(): 개별 군집의 중심점과 표준편차 제어 기능 추가
 - make_classification(): 노이즈를 포함한 데이터 만드는데 유용
- 군집화용 데이터 생성기: make_circle(),make_moon() API도 있음 - 군집화로 해결 어려울때

7.1_ K- 평균 알고리즘의 이해

군집화 알고리즘 테스트를 위한 데이터 생성

- 데이터 생성

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

- n_samples: 생성할 총 데이터의 개수
- n_features: 데이터의 피처 개수
- centers: 군집의 개수/개별 군집 중심점의 좌표
- cluster_std: 생성될 군집 데이터의 표준편차

```
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)
```

7.1_ K- 평균 알고리즘의 이해

군집화 알고리즘 테스트를 위한 데이터 생성

- 데이터 생성

【Output】

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

피처 데이터 세트 X는 200개의 레코드와 2개의 피처를 가지므로 shape은 (200, 2), 군집 타깃 데이터 세트인 y의 shape은 (200,). 그리고 3개의 cluster의 값은 [0, 1, 2]이며 각각 67, 67, 66개로 균일하게 구성돼 있습니다. 좀 더 데이터 가공을 편리하게 하기 위해서 위 데이터 세트를 DataFrame으로 변경하겠습니다. 피처의 이름은 ftr1, ftr2입니다.

```
import pandas as pd  
  
clusterDF = pd.DataFrame(data=X, columns=['ftr1', 'ftr2'])  
clusterDF['target'] = y  
clusterDF.head(3)
```

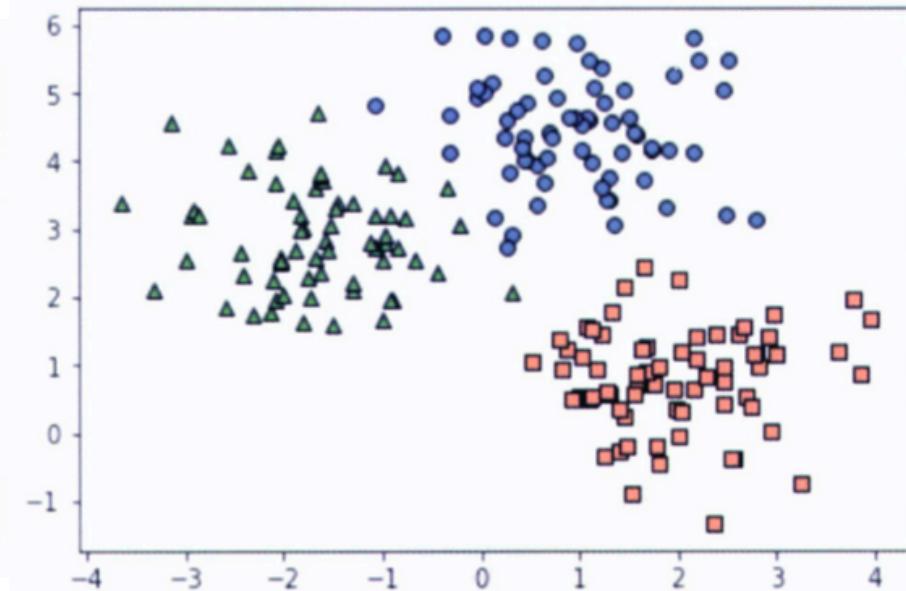
	ftr1	ftr2	target
0	-1.692427	3.622025	2
1	0.697940	4.428867	0
2	1.100228	4.606317	0

7.1_ K- 평균 알고리즘의 이해

군집화 알고리즘 테스트를 위한 데이터 생성

- make_blob() 군집화 분포

```
target_list = np.unique(y)
# 각 타깃별 산점도의 마커 값.
markers=['o', 's', '^', 'P', 'D', 'H', 'x']
# 3개의 군집 영역으로 구분한 데이터 세트를 생성했으므로 target_list는 [0, 1, 2]
# target==0, target==1, target==2 로 scatter plot을 marker별로 생성.
for target in target_list:
    target_cluster = clusterDF[clusterDF['target']==target]
    plt.scatter(x=target_cluster['ftr1'], y=target_cluster['ftr2'], edgecolor='k',
    marker=markers[target] )
plt.show()
```



7.1_ K- 평균 알고리즘의 이해

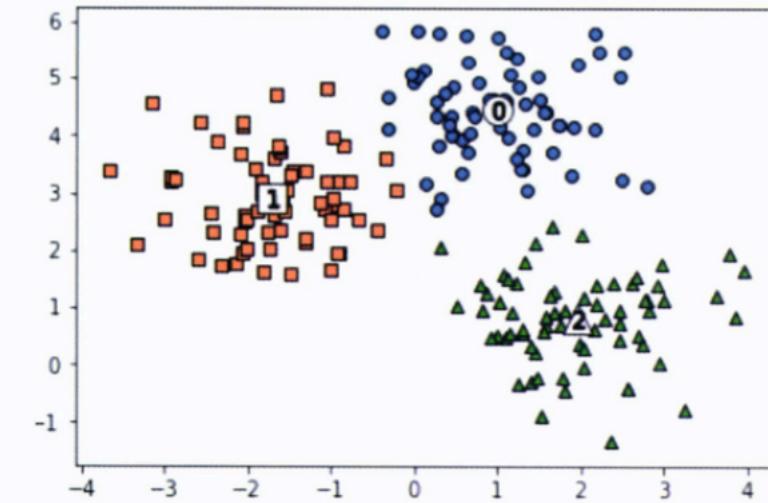
군집화 알고리즘 테스트를 위한 데이터 생성

• 군집화별 시각화

```
# KMeans 객체를 이용해 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()
```



7.1_ K- 평균 알고리즘의 이해

군집화 알고리즘 테스트를 위한 데이터 생성

- 결과

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

[Output]

target	kmeans_label	value
0	0	66
1	1	1
1	2	67
2	1	65
2	2	1

군집화 잘됨

아까랑 비교

VS

```
irisDF['target'] = iris.target
```

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

[Output]

```
target cluster
```

0	1	50
1	0	48
1	2	2
2	0	14
2	2	36

타겟1: 0,2로 군집

타겟2: 0,2로 군집(분산됨)

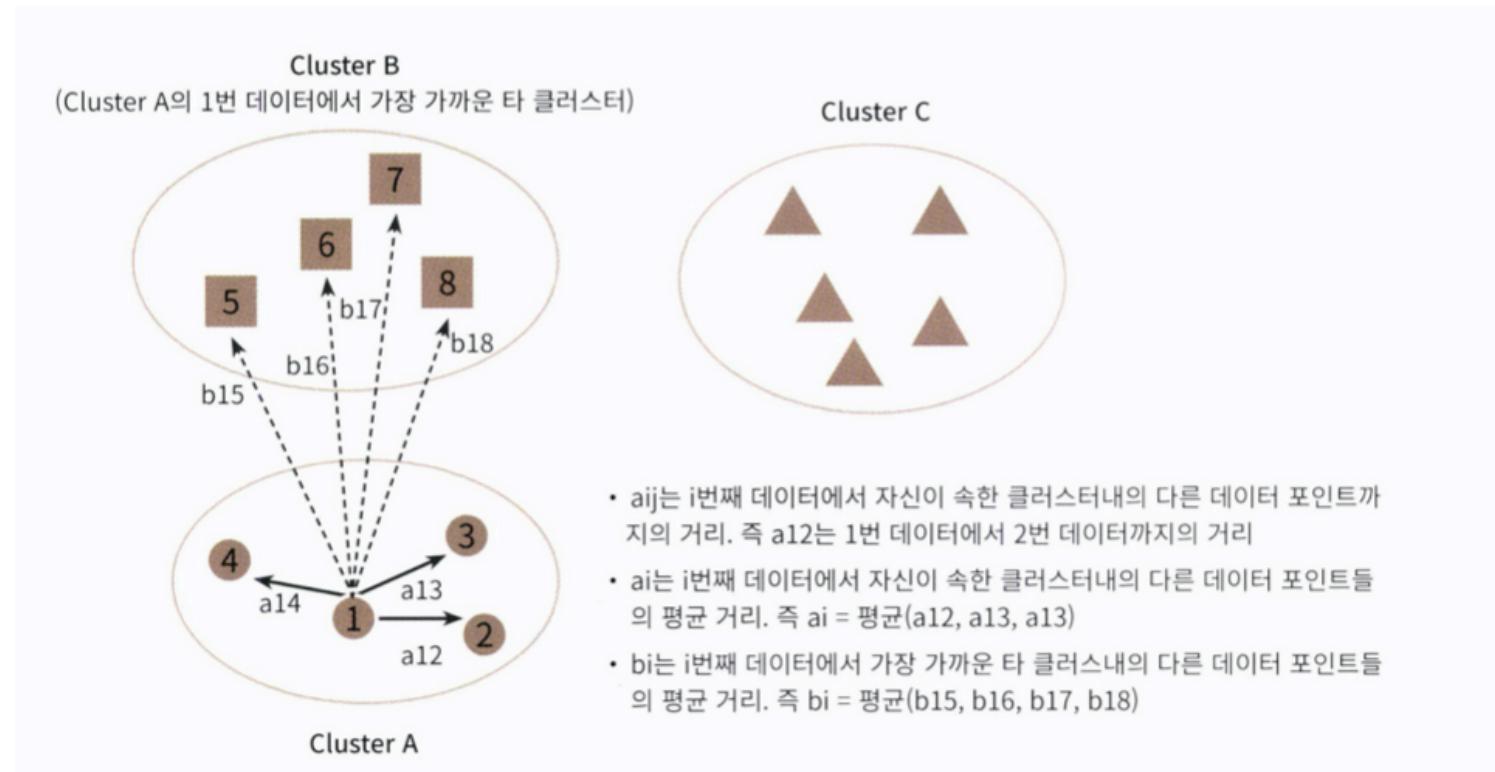
7.2_ 군집 평가(Cluster Evaluation)

- 군집화가 효율적으로 잘 됐는지 평가할 수 있는 지표
- 대부분의 경우 데이터의 군집화는 비교할만한 타깃 레이블을 가지고 있지 않음 (아까 붓꽃데이터는 가지고 있는 경우)
- 또한 분류가 유사해보여도 사실상 성격이 많이 다를 수 있음

7.2_ 군집 평가(Cluster Evaluation)

실루엣 분석의 개요

- 실루엣분석: 각 군집 간의 거리가 얼마나 효율적으로 분리돼 있는지 나타냄
- 효율적 분리= 군집간 거리는 멀고 & 동일 군집끼리는 가까움



7.2_ 군집 평가(Cluster Evaluation)

실루엣 분석의 개요

- 실루엣 계수: -1 ~ 1사이의 값을 가지며 1로 가까워질 수록 근처 군집과 더 멀리 있다는것

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

7.2_ 군집 평가(Cluster Evaluation)

실루엣 분석의 개요

- 실루엣 분석을 위한 메서드(사이킷런)
 - `sklearn.metrics.silhouette_samples(X, labels, metric='euclidean', **kwds)`: 인자로 X feature 데이터 세트와 각 피처 데이터 세트가 속한 군집 레이블 값인 labels 데이터를 입력해주면 각 데이터 포인트의 실루엣 계수를 계산해 반환합니다.
 - `sklearn.metrics.silhouette_score(X, labels, metric='euclidean', sample_size=None, **kwds)`: 인자로 X feature 데이터 세트와 각 피처 데이터 세트가 속한 군집 레이블 값인 labels 데이터를 입력해주면 전체 데이터의 실루엣 계수 값을 평균해 반환합니다. 즉, `np.mean(silhouette_samples())`입니다. 일반적으로 이 값이 높을수록 군집화가 어느 정도 잘 됐다고 판단할 수 있습니다. 하지만 무조건 이 값이 높다고 해서 군집화가 잘 됐다고 판단할 수는 없습니다.

7.2_ 군집 평가(Cluster Evaluation)

실루엣 분석의 개요

- 좋은 군집화가 되려면:

- 실루엣 계수 평균값: 0~1사이의 값을 가지며 1에 가까울 수록 굿
- 개별 군집의 평균값의 편차는 작아야함 = 개별 군집의 실루엣 평균값이 전체 실루엣 계수의 평균값에 크게 벗어나면 안됨

7.2_ 군집 평가(Cluster Evaluation)

붓꽃 데이터 세트를 이용한 군집 평가

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

# 실루엣 분석 평가 지표 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_

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

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

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

7.2_ 군집 평가(Cluster Evaluation)

붓꽃 데이터 세트를 이용한 군집 평가

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

붓꽃 데이터 세트 Silhouette Analysis Score:0.553

평균 실루엣 계수

	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

7.2_ 군집 평가(Cluster Evaluation)

붓꽃 데이터 세트를 이용한 군집 평가

- group_by를 이용한 silhouette_coeff칼럼의 평균값

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

【Output】

cluster
0 0.417182
1 0.797630
2 0.451105

평균값의 차이가 너무 큼

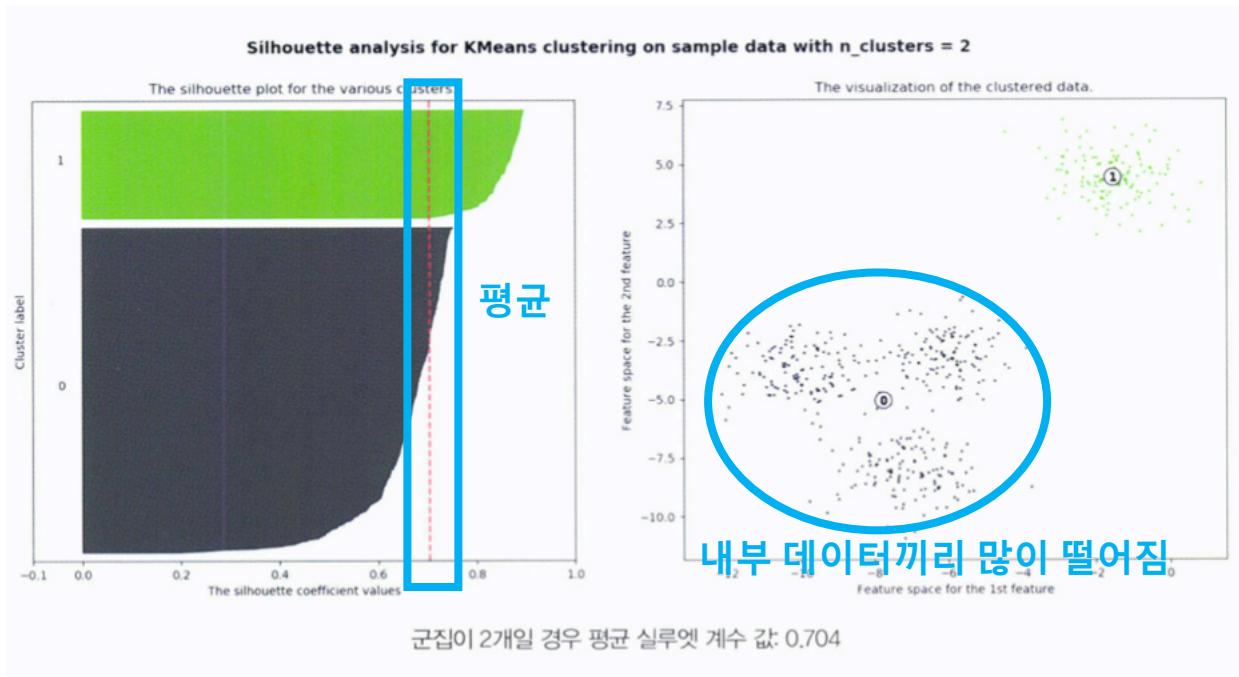


- 좋은 군집화가 되려면:
 - 실루엣 계수 평균값: 0~1사이의 값을 가지며 1에 가까울 수록 굿
 - 개별 군집의 평균값의 편차는 작아야함 = 개별 군집의 실루엣 평균값이 전체 실루엣 계수의 평균값에 크게 벗어나면 안됨

7.2_ 군집 평가(Cluster Evaluation)

군집별 평균 실루엣 계수의 시각화를 통한 군집 개수 최적화 방법

- 잘 된 군집화 = 군집별로 적당히 분리된 거리를 유지하면서도 군집내의 데이터가 서로 뭉쳐 있는 경우

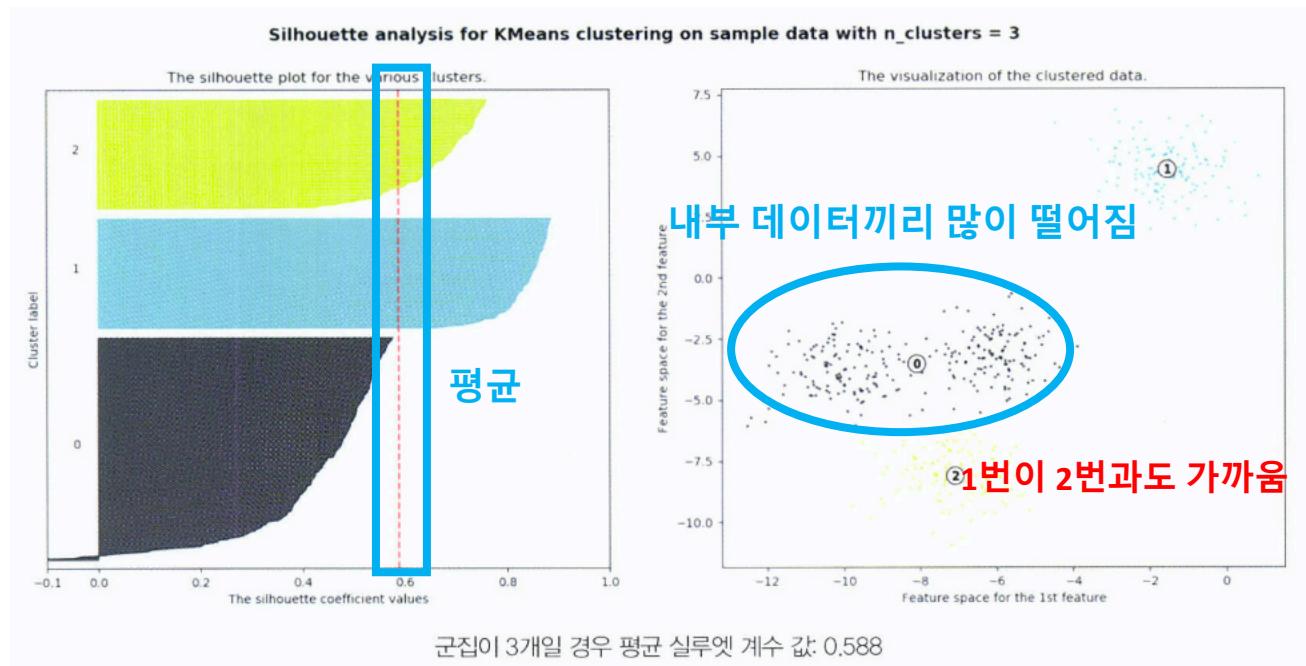


- x축: 실루엣 계수 값
- y축: 개별 군집과 이에 속하는 데이터

7.2_ 군집 평가(Cluster Evaluation)

군집별 평균 실루엣 계수의 시각화를 통한 군집 개수 최적화 방법

- 군집 개수가 3개인 경우

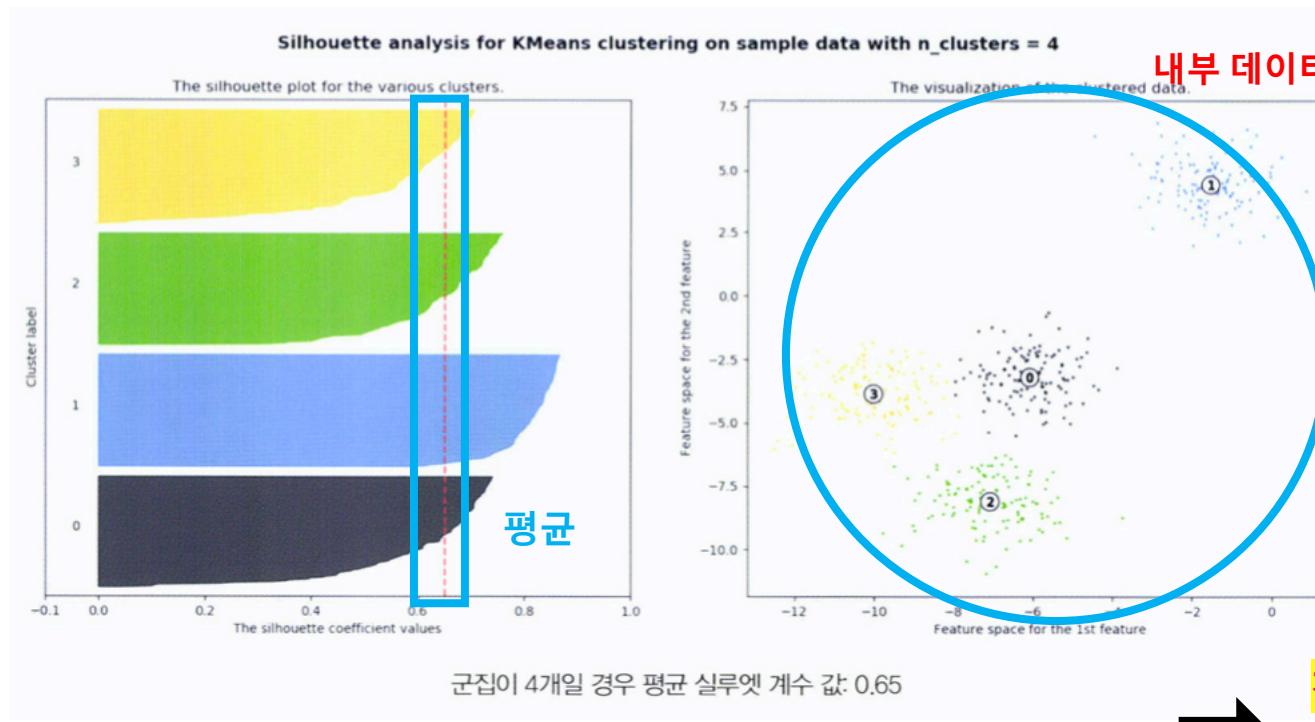


- 1,2번: 평균보다 높은 실루엣 계수
- 0번: 평균보다 낮은 실루엣 계수

7.2_ 군집 평가(Cluster Evaluation)

군집별 평균 실루엣 계수의 시각화를 통한 군집 개수 최적화 방법

- 군집 개수가 4개인 경우



- 1번: 평균보다 높은 실루엣 계수
- 0,2번: 절반 이상이 평균보다 높은 실루엣 계수
- 3번: 1/3정도가 평균보다 높은 실루엣 계수

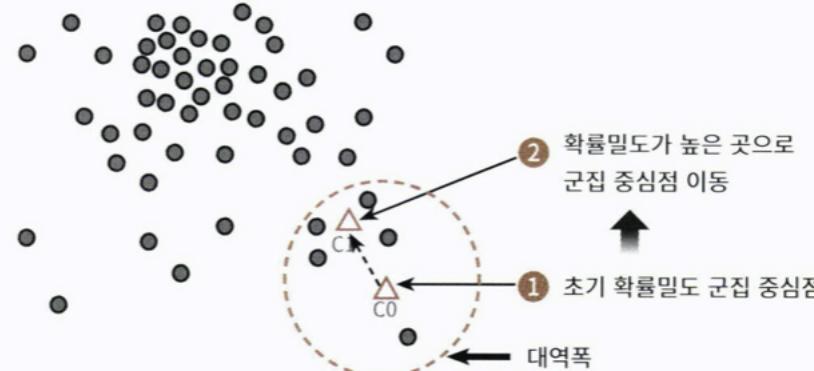
7.3_ 평균 이동(Mean Shift)

평균이동(Mean Shift)의 개요

- 평균 이동: k -평균과 유사하게 중심을 군집의 중심으로 지속적으로 움직이면서 군집화 수행

평균이동: 데이터가 많이 모인 밀도 높은 곳으로 이동

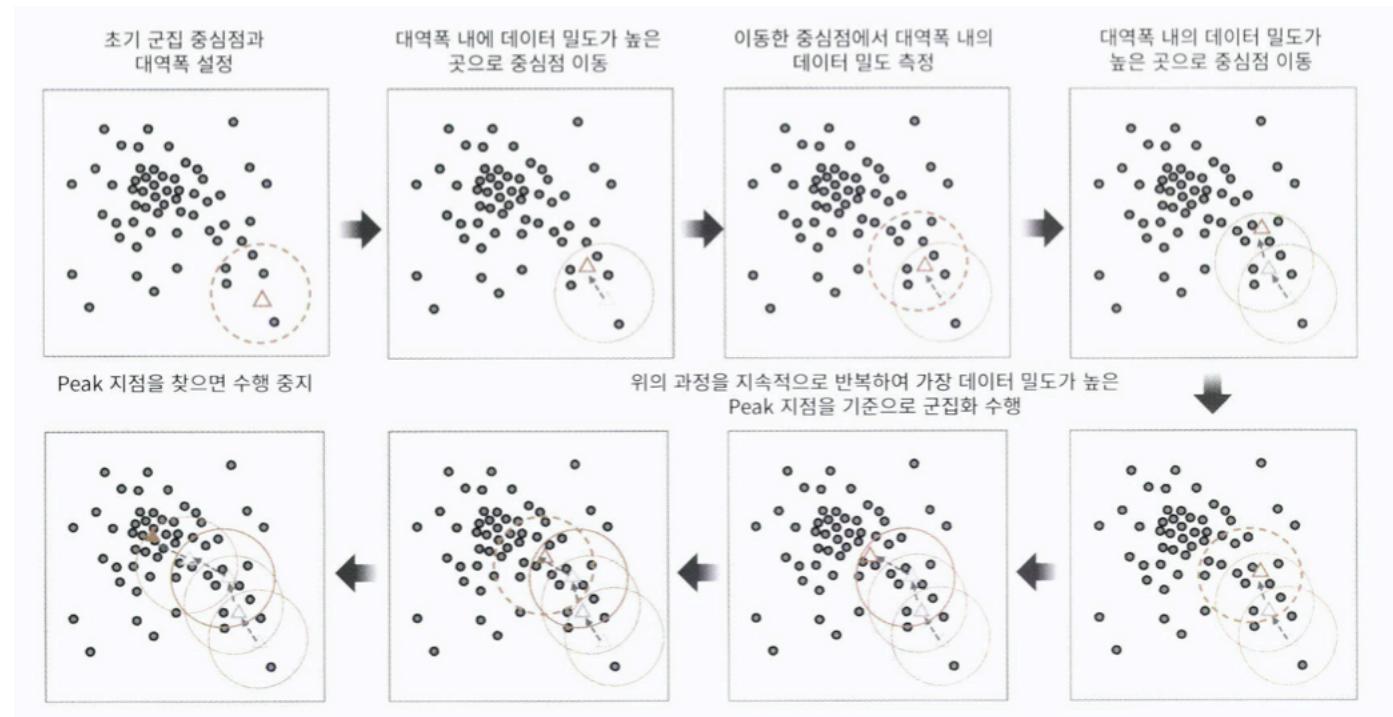
k -평균: 데이터 평균 거리 중심으로 이동



7.3_ 평균 이동(Mean Shift)

평균이동(Mean Shift)의 개요

- 데이터 분포도를 이용해 군집의 중심점을 찾음 = 확률밀도 함수 이용
- 데이터가 가장 많이 모여 있는 곳(피크포인트)을 중심점으로 선정
= KDE 모델 이용
 - > (피크포인트 찾을 때까지)
반복적으로 수행



7.3_ 평균 이동(Mean Shift)

평균이동(Mean Shift)의 개요

- 평균이동 알고리즘 적용

K-평균과 다르게 군집의 개수를 지정할 필요가 없음

- MeanShift 클래스 사용

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

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

meanshift= MeanShift(bandwidth=0.9)
cluster_labels = meanshift.fit_predict(X)
print('cluster labels 유형:', np.unique(cluster_labels))
```

7.3_ 평균 이동(Mean Shift)

평균이동(Mean Shift)의 개요

[Output]

cluster labels 유형: [0 1 2 3 4 5 6 7]



Bandwidth을 작게 할수록 군집 개수가 많아짐

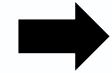
군집이 8개로 분류됨

- 평균이동 알고리즘 적용_(bandwidth조금 크게해서)MeanShift 수행

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

[Output]

군집이 3개로 분류됨



Good

cluster labels 유형: [0 1 2]

7.3_ 평균 이동(Mean Shift)

평균이동(Mean Shift)의 개요

- 평균이동 알고리즘 적용_최적화 된 bandwidth 찾기 = `estimate_bandwidth()`

```
from sklearn.cluster import estimate_bandwidth  
  
bandwidth = estimate_bandwidth(X, quantile=0.2)  
print('bandwidth 값:', round(bandwidth, 3))
```

【Output】

```
bandwidth 값: 1.444
```

`estimate_bandwidth()`로 측정된 bandwidth를 평균 이동 입력 값으로 적용해 동일한 `make_blobs()` 데이터 세트에 군집화를 수행해 보겠습니다.

```
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, quantile=0.2)  
  
meanshift = MeanShift(bandwidth=1)  
cluster_labels = meanshift.fit_predict(X)  
print('cluster labels 유형:', np.unique(cluster_labels))
```

【Output】

```
cluster labels 유형: [1 0 2]
```

군집이 3개로 분류됨

7.3_ 평균 이동(Mean Shift)

평균이동(Mean Shift)의 개요

- 평균이동 알고리즘 적용_시각화= **cluster_centers_** 속성 이용

```
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', '*']

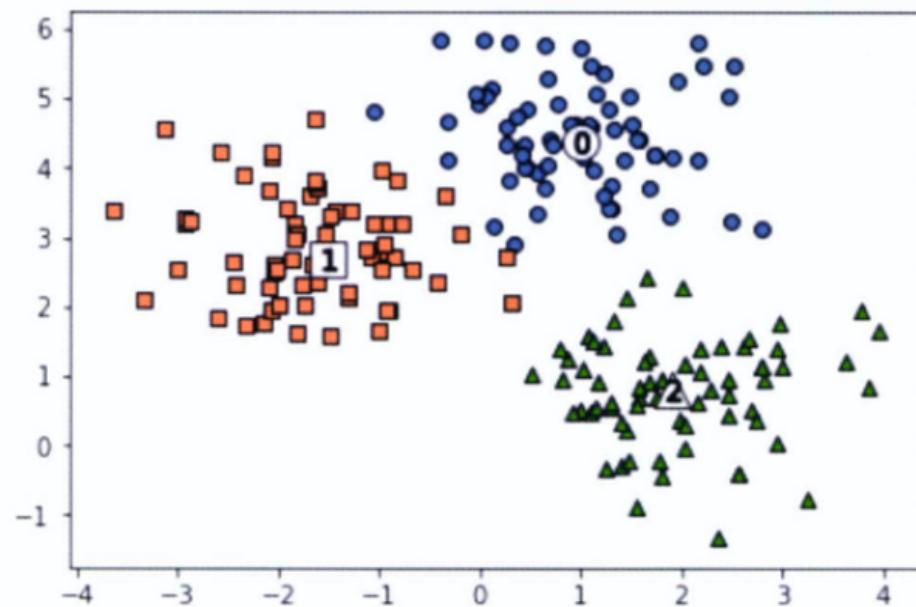
for label in unique_labels:
    label_cluster = clusterDF[clusterDF['meanshift_label']==label]
    center_x_y = centers[label]
    # 군집별로 다른 marker로 scatter plot 적용
    plt.scatter(x=label_cluster['ftr1'], y=label_cluster['ftr2'], edgecolor='k',
                marker=markers[label] )
```

```
# 군집별 중심 시각화
plt.scatter(x=center_x_y[0], y=center_x_y[1], s=200, color='white',
            edgecolor='k', 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()
```

7.3_ 평균 이동(Mean Shift)

평균이동(Mean Shift)의 개요

- 평균이동 알고리즘 적용_시각화= `cluster_centers_` 속성 이용



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

[Output]

target	meanshift_label	value_counts()
0	0	66
0	1	1
1	2	67
2	1	66

타겟 = 라벨 → Good

평균이동:

- 데이터 세트의 형태를 특정 형태로 가정한다든가, 특정 분포도 기반의 모델로 가정하지 않기 때문에 유연한 군집화 가능
- 이상치의 영향을 크게 받지 않음
- 미리 군집의 개수를 정할 필요도 없음

7.3_ 평균 이동(Mean Shift)

평균이동(Mean Shift)의 개요

- 평균이동 알고리즘

장점

- 데이터 세트의 형태를 특정 형태로 가정한다든가, 특정 분포도 기반의 모델로 가장하지 않기 때문에 유연한 군집화 가능
- 이상치의 영향을 크게 받지 않음
- 미리 군집의 개수를 정할 필요도 없음



단점

- Bandwidth() 크기에 따른 영향이 크다
- 알고리즘 수행시간이 오래 걸린다