

Chapter 29. Clustering



비지도 학습

얀 르쿤



'인공지능 4대 선구자'로 꼽히는 얀 르쿤, 제프리 힌턴, 요수아 벤지오, 앤드류 응(왼쪽부터). [자료=KAIST]

- 지능이 케이크라면, 비지도 학습은 케이크의 빵, 지도 학습은 케이크 위의 크림, 강화학습은 케이크위의 체리
- 이 말은 비지도학습의 무궁무진한 가능성에 대한 언급

비지도 학습

비지도 학습

- 군집 Clustering : 비슷한 샘플을 모음
- 이상치 탐지 Outlier detection : 정상 데이터가 어떻게 보이는지 학습, 비정상 샘플을 감지
- 밀도 추정 : 데이터셋의 확률 밀도 함수 Probability Density Function PDF를 추정. 이상치 탐지 등에 사용

비지도 학습

K-Means

- 군집화에서 가장 일반적인 알고리즘
- 군집 중심(centroid)이라는 임의의 지점을 선택해서 해당 중심에 가장 가까운 포인트들을 선택하는 군집화
- 일반적인 군집화에서 가장 많이 사용되는 기법
- 거리 기반 알고리즘으로 속성의 개수가 매우 많을 경우 군집화의 정확도가 떨어짐

비지도 학습

K-Means 알고리즘

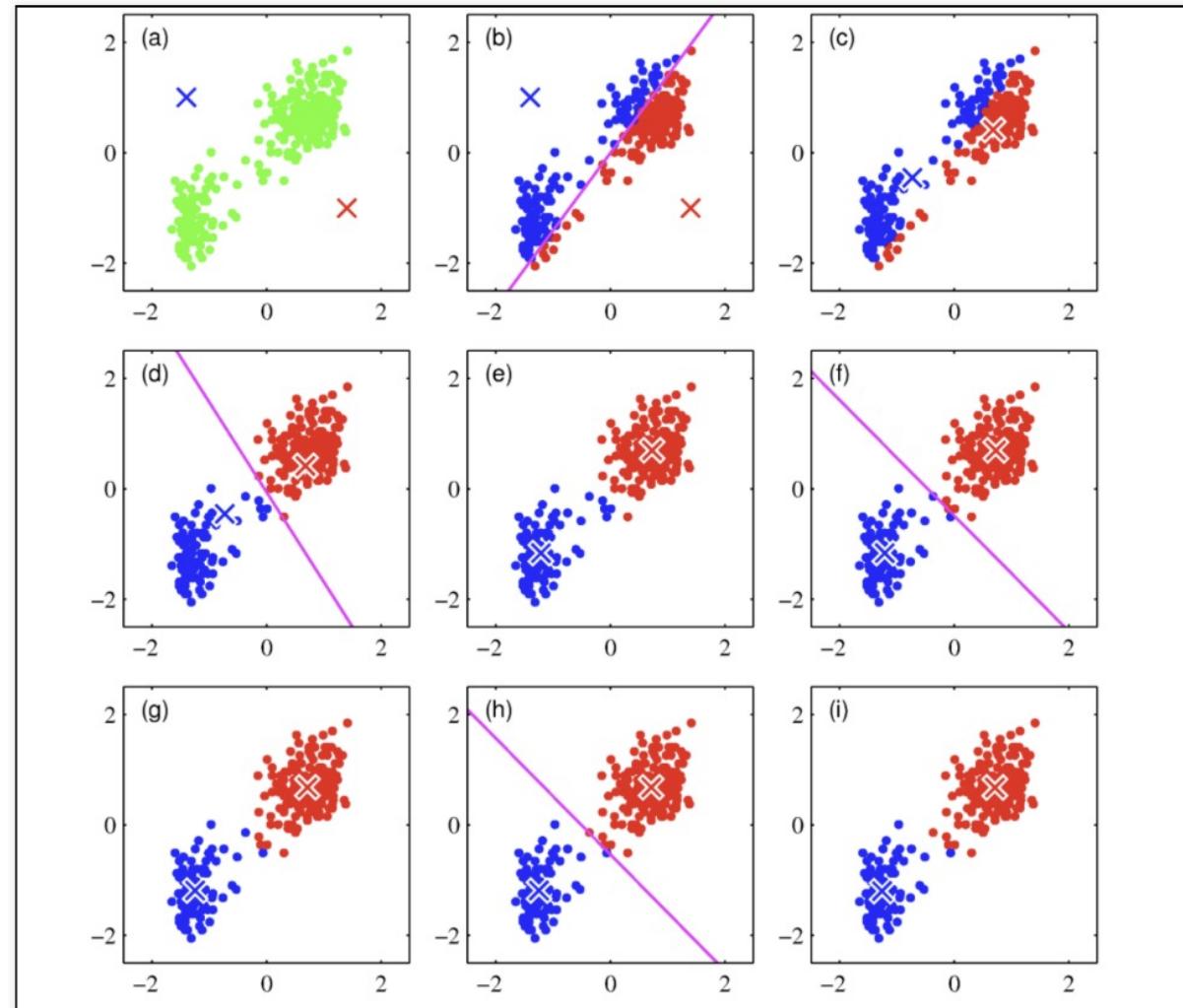
입력: 훈련집합 $\mathbb{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$, 군집의 개수 k

출력: 군집집합 $C = \{c_1, c_2, \dots, c_k\}$

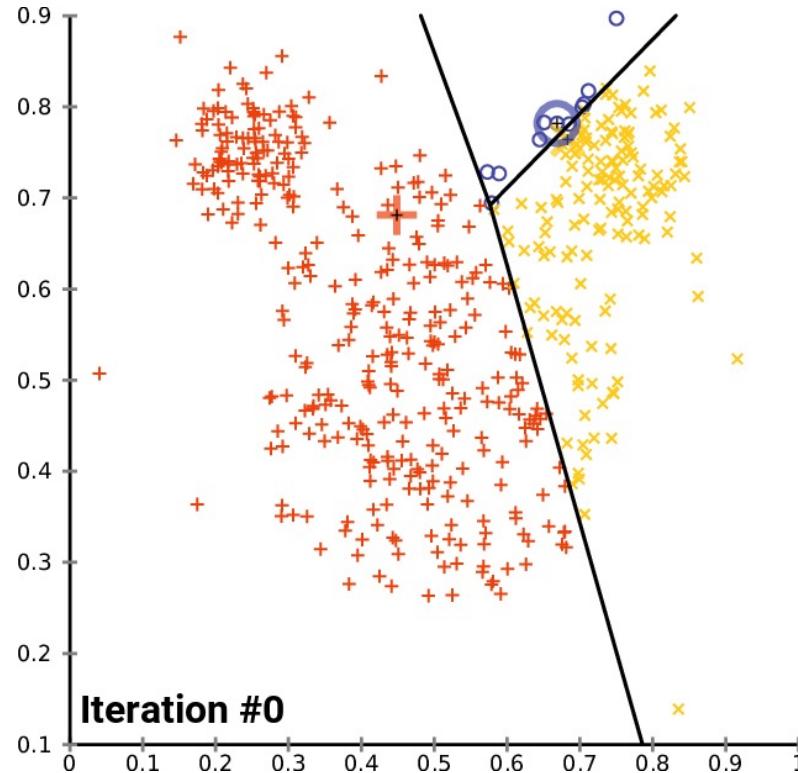
- 1 k 개의 군집 중심 $Z = \{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_k\}$ 를 초기화한다.
- 2 while (true)
 - 3 for ($i=1$ to n)
 - 4 \mathbf{x}_i 를 가장 가까운 군집 중심에 배정한다.
 - 5 if (라인 3~4에서 이루어진 배정이 이전 루프에서의 배정과 같으면) break
 - 6 for ($j=1$ to k)
 - 7 \mathbf{z}_j 에 배정된 샘플의 평균으로 \mathbf{z}_j 를 대치한다.
 - 8 for ($j=1$ to k)
 - 9 \mathbf{z}_j 에 배정된 샘플을 c_j 에 대입한다.

비지도 학습

원리



비지도 학습



- 초기 중심점을 설정
- 각 데이터는 가장 가까운 중심점에 소속
- 중심점에 할당된 평균값으로 중심점 이동
- 각 데이터는 이동된 중심점 기준으로 가장 가까운 중심점에 소속
- 다시 중심점에 할당된 데이터들의 평균값으로 중심점 이동
- 데이터들의 중심점 소속 변경이 없으면 종료

비지도 학습

iris 데이터로 실습

```
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()
```

특징 이름 - 항상 뒤에 (cm)가 불편했다

```
| iris.feature_names
```

```
[ 'sepal length (cm)',  
  'sepal width (cm)',  
  'petal length (cm)',  
  'petal width (cm)' ]
```

비지도 학습

뒷 글자 자르기~

```
| cols = [each[:-5] for each in iris.feature_names]
| cols
[ 'sepal length', 'sepal width', 'petal length', 'petal width' ]
```

비지도 학습

iris 데이터 정리

```
iris_df = pd.DataFrame(data=iris.data, columns=cols)
iris_df.head()
```

| | 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 |
| 3 | 4.6 | 3.1 | 1.5 | 0.2 |
| 4 | 5.0 | 3.6 | 1.4 | 0.2 |

비지도 학습

편의상 두 개의 특성만

```
feature = iris_df[ ['petal length','petal width']]  
feature.head()
```

| | petal length | petal width |
|---|--------------|-------------|
| 0 | 1.4 | 0.2 |
| 1 | 1.4 | 0.2 |
| 2 | 1.3 | 0.2 |
| 3 | 1.5 | 0.2 |
| 4 | 1.4 | 0.2 |

비지도 학습

군집화 시작~

```
| model = KMeans(n_clusters=3)
|     model.fit(feature)
|
| KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
|         n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
|         random_state=None, tol=0.0001, verbose=0)
```

- `n_clusters` : 군집화 할 개수, 즉 군집 중심점의 개수
- `init` : 초기 군집 중심점의 좌표를 설정하는 방식을 결정
- `max_iter` : 최대 반복 횟수, 모든 데이터의 중심점 이동이 없으면 종료

비지도 학습

결과 라벨~ (군집화라서 지도학습의 라벨과 다르다)

model.labels_

비지도 학습

군집 중심값

```
| model.cluster_centers_  
  
array([[1.462      ,  0.246      ],  
       [4.26923077,  1.34230769],  
       [5.59583333,  2.0375     ]])
```

비지도 학습

다시 정리 (그림 그리기 위해)

```
predict = pd.DataFrame(model.predict(feature), columns=['cluster'])
feature = pd.concat([feature, predict], axis=1)
feature.head()
```

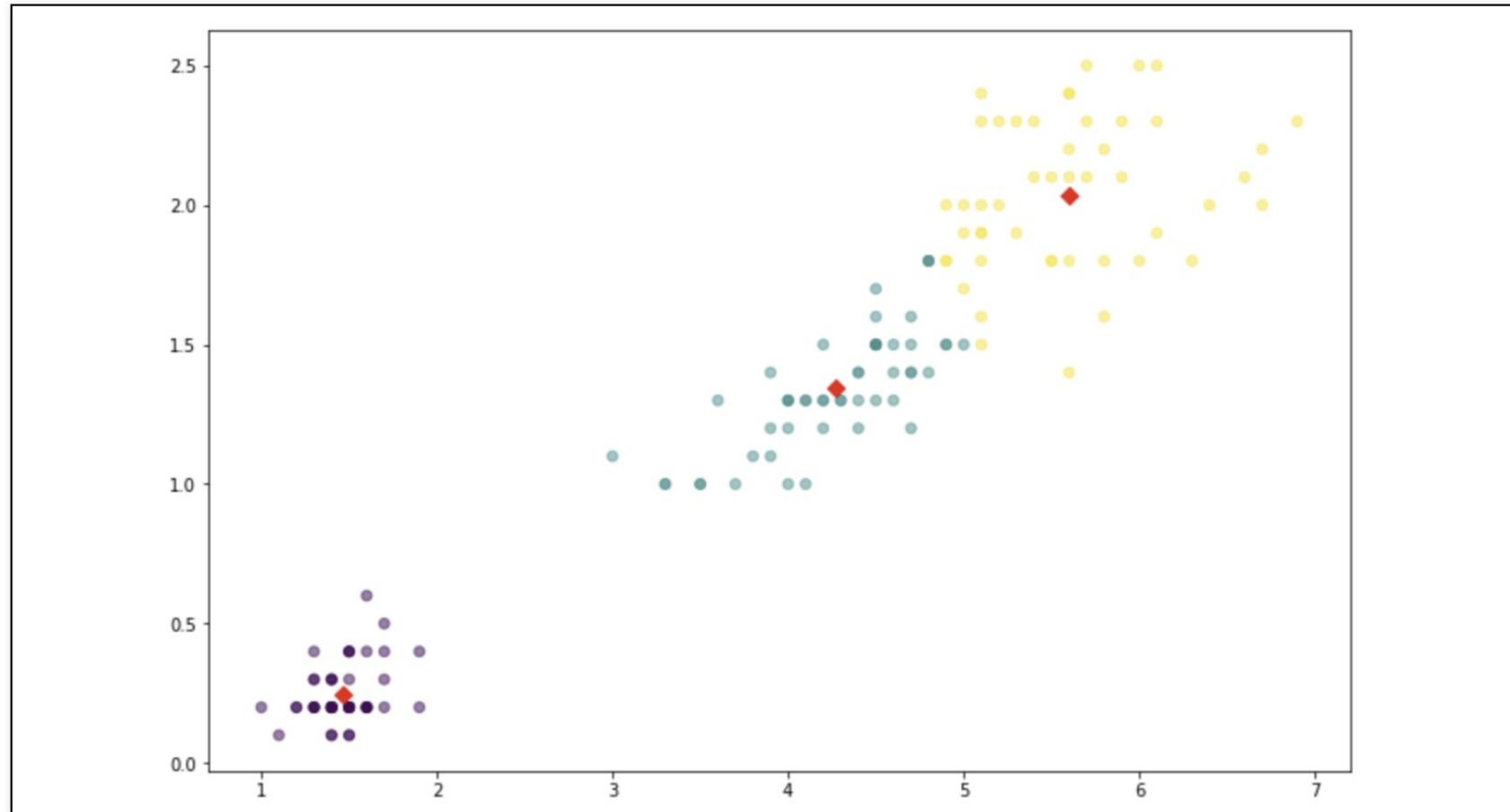
| | petal length | petal width | cluster |
|---|--------------|-------------|---------|
| 0 | 1.4 | 0.2 | 0 |
| 1 | 1.4 | 0.2 | 0 |
| 2 | 1.3 | 0.2 | 0 |
| 3 | 1.5 | 0.2 | 0 |
| 4 | 1.4 | 0.2 | 0 |

결과를 확인하기 위해

```
centers = pd.DataFrame(model.cluster_centers_,  
                        columns=['petal length','petal width'])  
center_x = centers['petal length']  
center_y = centers['petal width']  
  
plt.figure(figsize=(12,8))  
plt.scatter(feature['petal length'], feature['petal width'],  
            c=feature['cluster'], alpha=0.5)  
plt.scatter(center_x,center_y,s=50,marker='D',c='r')  
plt.show()
```

비지도 학습

결론



make_blobs

make_blobs

make_blobs

```
| from sklearn.datasets import make_blobs  
  
X, y = make_blobs(n_samples=200, n_features=2, centers=3,  
                   cluster_std=0.8, random_state=0)  
print(X.shape, y.shape)  
  
unique, counts = np.unique(y, return_counts=True)  
print(unique,counts)
```

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

- 군집화 연습을 위한 데이터 생성기

make_blobs

데이터 정리

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

| | ftr1 | ftr2 | target |
|---|-----------|----------|--------|
| 0 | -1.692427 | 3.622025 | 2 |
| 1 | 0.697940 | 4.428867 | 0 |
| 2 | 1.100228 | 4.606317 | 0 |
| 3 | -1.448724 | 3.384245 | 2 |
| 4 | 1.214861 | 5.364896 | 0 |

make_blobs

군집화

```
kmeans = KMeans(n_clusters=3, init='k-means++', max_iter=200, random_state=13)
cluster_labels = kmeans.fit_predict(X)
cluster_df['kmeans_label'] = cluster_labels
```

make_blobs

결과 도식화

```
centers = kmeans.cluster_centers_
unique_labels = np.unique(cluster_labels)
markers=['o', 's', '^', 'P', 'D', 'H', 'x']

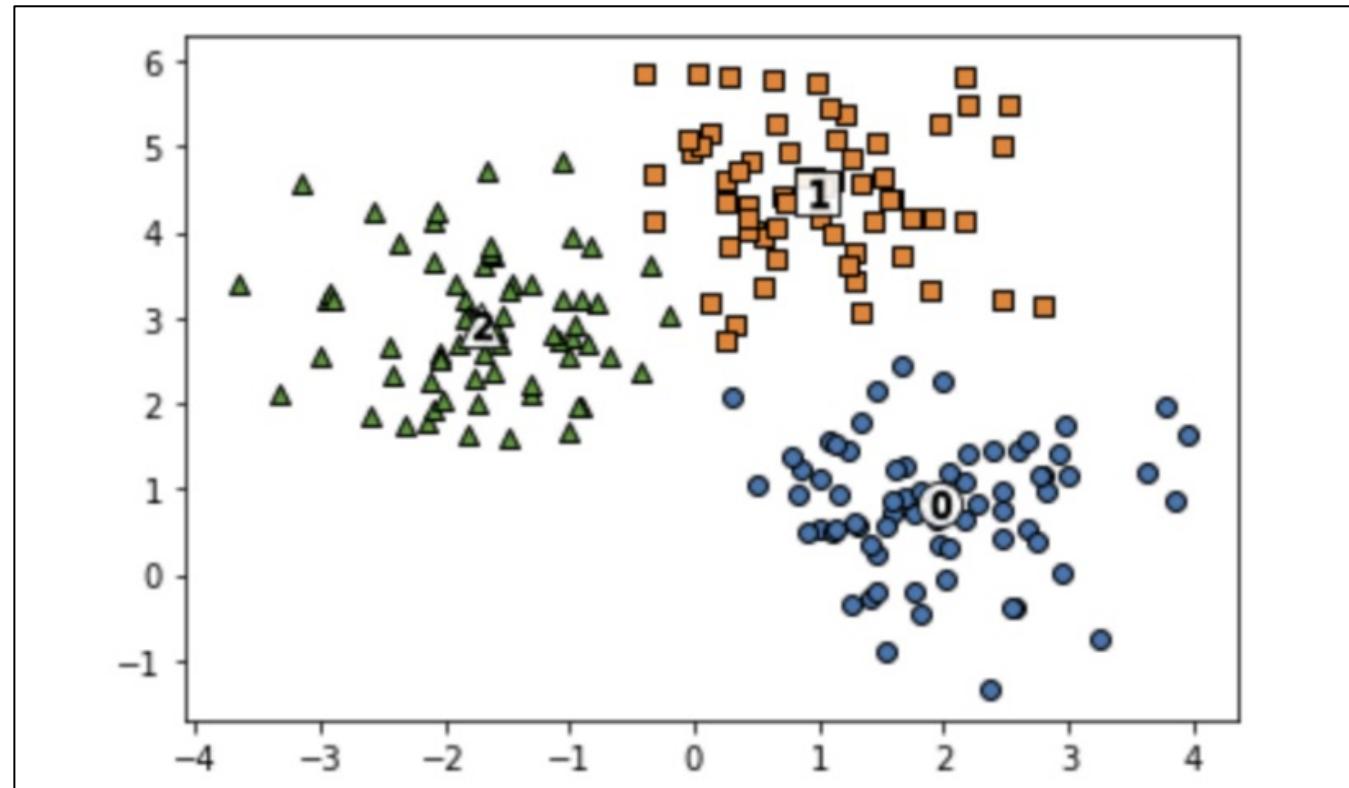
for label in unique_labels:
    label_cluster = cluster_df[cluster_df['kmeans_label']==label]
    center_x_y = centers[label]
    plt.scatter(x=label_cluster['ftr1'], y=label_cluster['ftr2'], edgecolor='k',
                marker=markers[label])

    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()
```

make_blobs

결론



make_blobs

결과 확인

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

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

Name: kmeans_label, dtype: int64

군집 평가

군집 평가

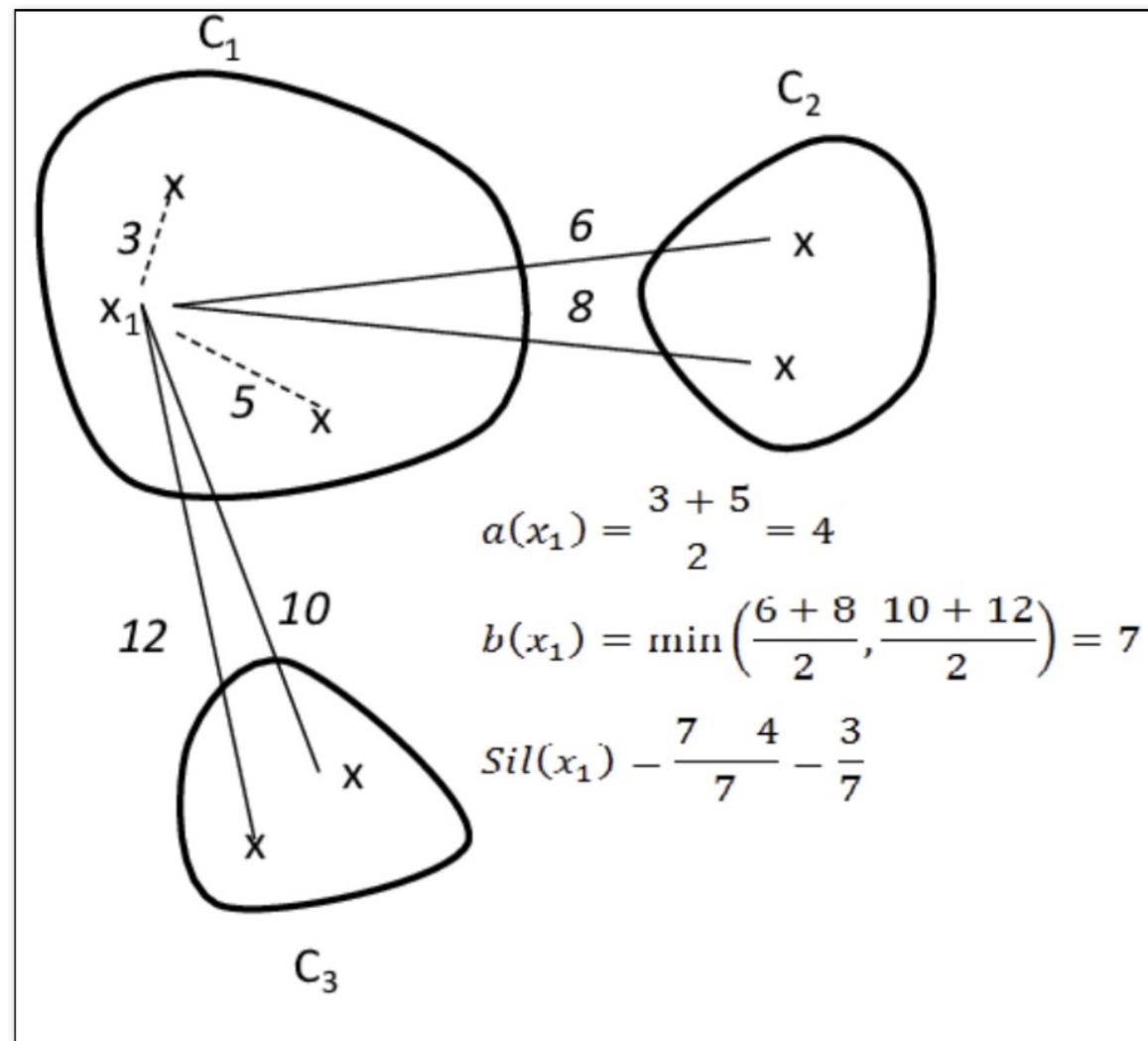
군집 결과의 평가

- 분류기는 평가 기준(정답)을 가지고 있지만, 군집은 그렇지 않다.
- 군집 결과를 평가하기 위해 실루엣 분석을 많이 활용한다.

실루엣 분석

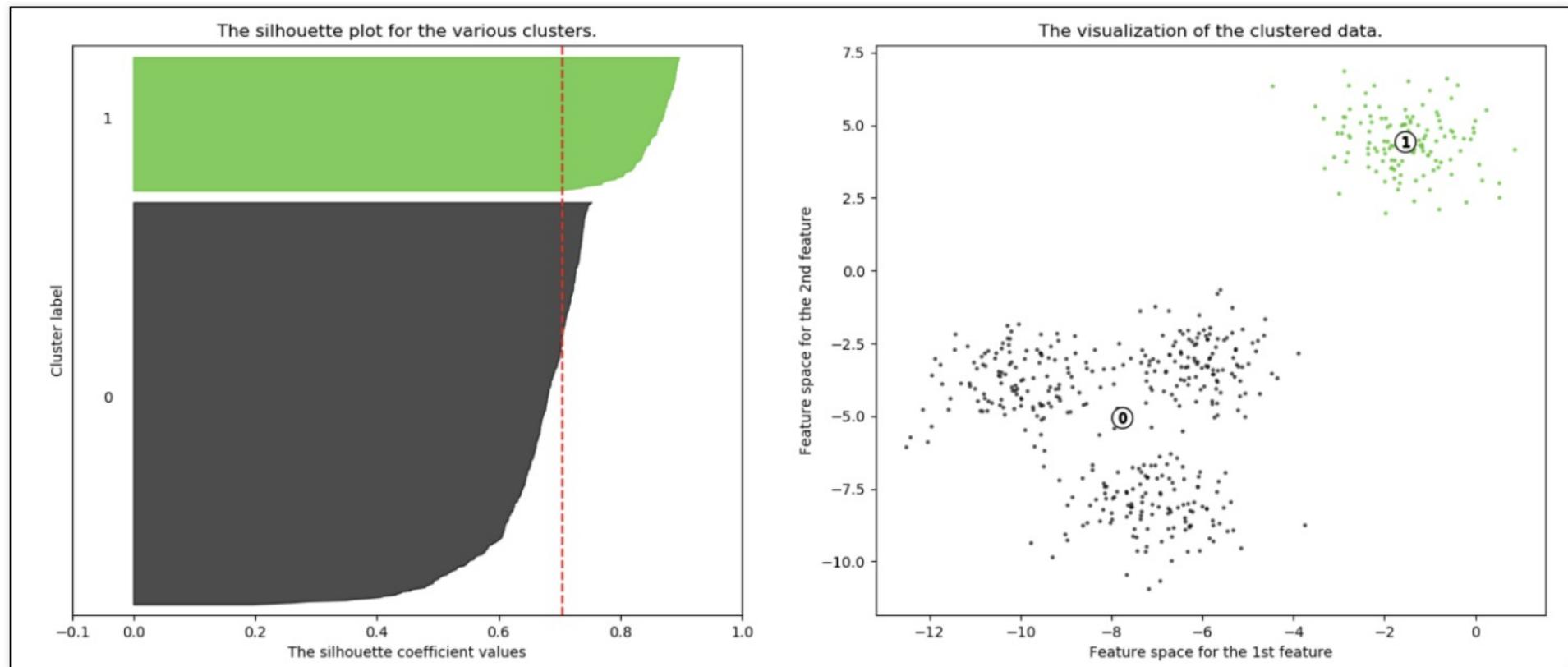
- 실루엣 분석은 각 군집 간의 거리가 얼마나 효율적으로 분리되어 있는지 나타냄
- 다른 군집과는 거리가 떨어져 있고, 동일 군집간의 데이터는 서로 가깝게 잘 뭉쳐 있는지 확인
- 군집화가 잘 되어 있을 수록 개별 군집은 비슷한 정도의 여유공간을 가지고 있음
- 실루엣 계수 : 개별 데이터가 가지는 군집화 지표

군집 평가



군집 평가

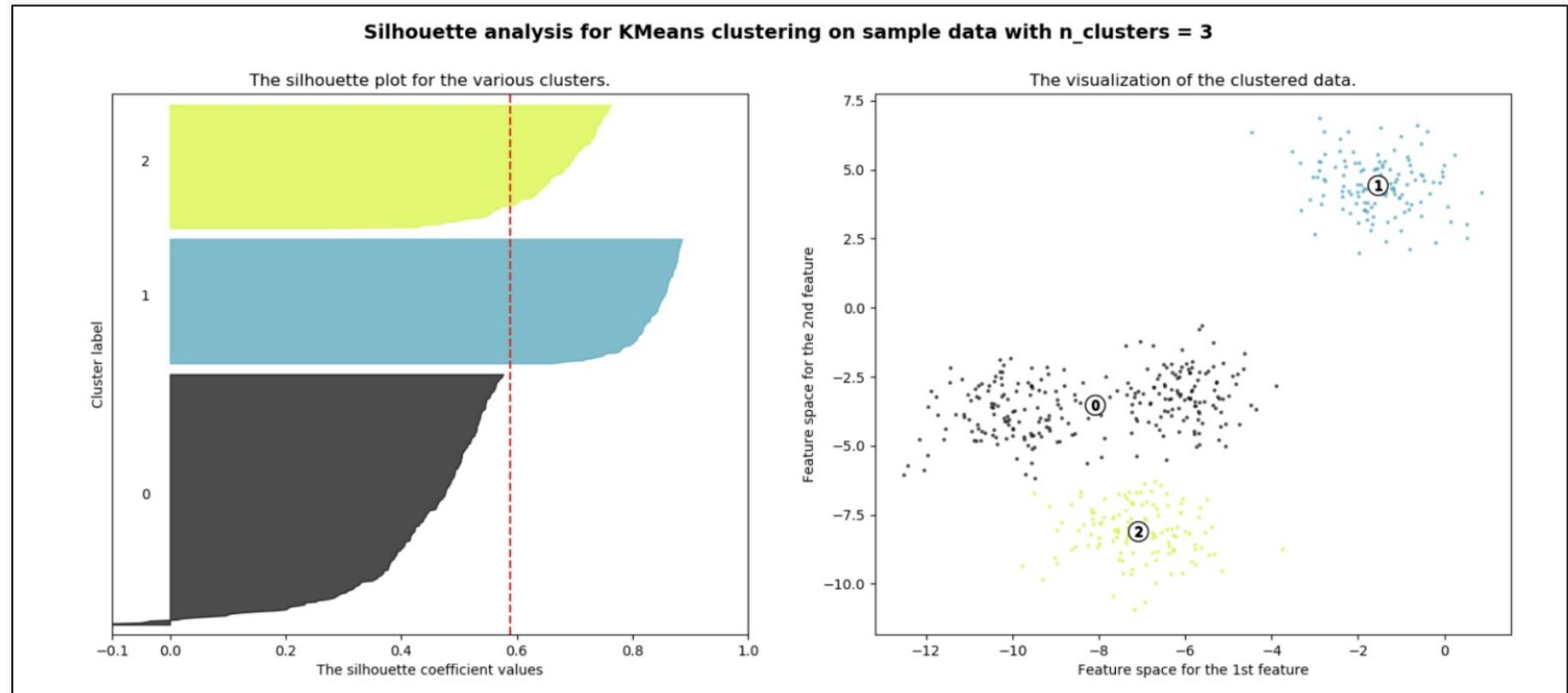
직접 보면서 이해해 보자 n=2인 경우



- 1번 군집의 경우 0번 군집과 떨어져 있고 잘 뭉쳐 있음
- 0번 군집의 경우 내부 데이터끼리 많이 떨어져 있음

군집 평가

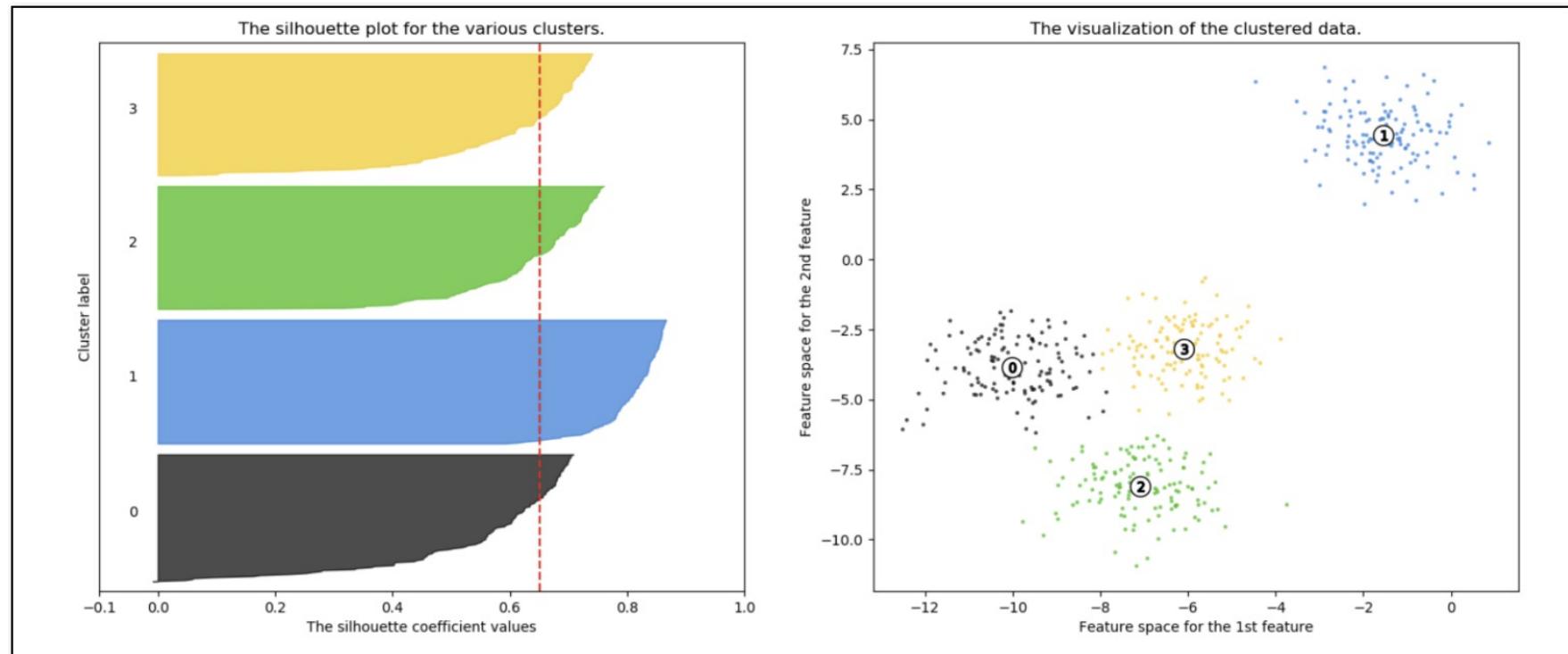
n=3인 경우



- 0번 군집의 경우 2번 군집과 가깝게 위치해 있음

군집 평가

n=4인 경우



데이터 읽고

```
from sklearn.datasets import load_iris  
from sklearn.cluster import KMeans  
import pandas as pd  
  
iris = load_iris()  
feature_names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']  
iris_df = pd.DataFrame(data=iris.data, columns=feature_names)  
kmeans = KMeans(n_clusters=3, init='k-means++', max_iter=300,  
                 random_state=0).fit(iris_df)
```

군집 평가

군집 결과 정리하고

```
| iris_df['cluster'] = kmeans.labels_
| iris_df.head()
```

| | 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 |

군집 평가

군집 결과 평가를 위한 작업

```
| from sklearn.metrics import silhouette_samples, silhouette_score  
  
avg_value = silhouette_score(iris.data, iris_df['cluster'])  
score_values = silhouette_samples(iris.data, iris_df['cluster'])  
  
print('avg_value' , avg_value)  
print('silhouette_samples( ) return 값의 shape' , score_values.shape)  
  
avg_value 0.5528190123564091  
silhouette_samples( ) return 값의 shape (150, )
```

군집 평가

yellowbrick을 설치하고

```
!pip install yellowbrick
```

✓ 1.0s

```
Collecting yellowbrick
```

```
  Downloading yellowbrick-1.3.post1-py3-none-any.whl (271 kB)
```

```
 |██████████| 271 kB 4.9 MB/s
```

```
Requirement already satisfied: numpy<1.20,>=1.16.0 in  
/Users/pw/miniforge3/envs/tf25/lib/python3.8/site-packages (from yellowbrick) (1.19.5)
```

실루엣 플랏의 결과

```
from yellowbrick.cluster import silhouette_visualizer  
silhouette_visualizer(kmeans, iris.data, colors='yellowbrick')
```

✓ 0.1s

Python

