

Clustering

PinkLAB Edu

11 November, 2025

Table of Contents

1	비지도 학습	4
1.1	얀르쿤	4
1.2	비지도 학습	4
1.3	K-Means.....	4
1.4	K-Means 알고리즘.....	5
1.5	원리.....	6
1.6	iris 데이터로 실습.....	7
1.7	특징 이름 - 항상 뒤에 (cm)가 붙편했다.....	8
1.8	뒷 글자 자르기~	8
1.9	iris 데이터 정리	8
1.10	편의상 두 개의 특성만.....	9
1.11	군집화 시작~	9
1.12	결과 라벨~ (군집화라서 지도학습의 라벨과 다르다)	9
1.13	군집 중심값	10
1.14	다시 정리 (그림 그리기 위해)	10
1.15	결과를 확인하기 위해	10
1.16	결론.....	11
2	make_blobs	12
2.1	make_blobs	12
2.2	데이터 정리	12
2.3	군집화	12
2.4	결과 도식화	13
2.5	결론.....	14
2.6	결과 확인.....	14
3	군집 평가.....	15
3.1	군집 결과의 평가	15
3.2	실루엣 분석	15
3.3	직접 보면서 이해해 보자 n=2인 경우	16
3.4	n=3인 경우	16

3.5 n=4인 경우	17
3.6 데이터 읽고	17
3.7 군집 결과 정리하고	17
3.8 군집 결과 평가를 위한 작업.....	18
3.9 실루엣 점수를 시각화	18
3.10 실루엣 플랏의 결과	19

1 비지도 학습

1.1 얀르쿤



'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

1.2 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

1.3 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

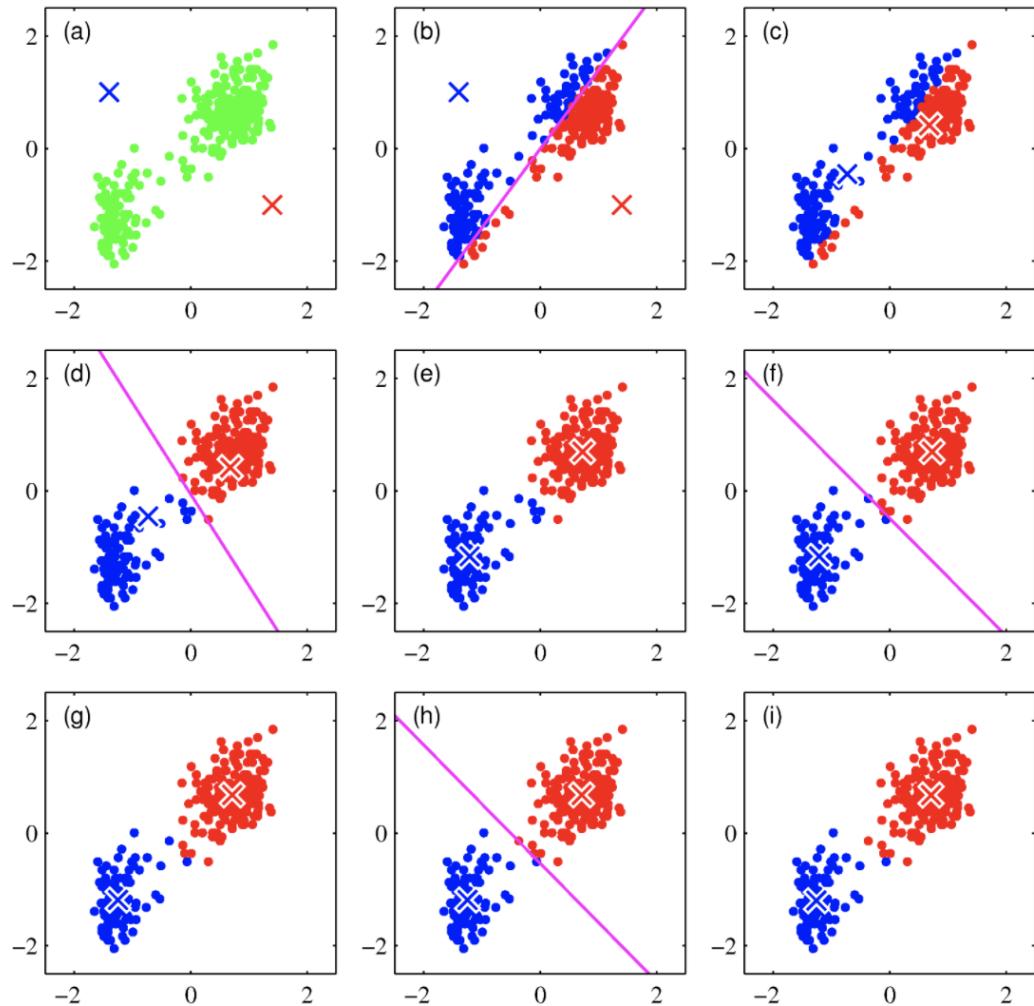
1.4 K-Means 알고리즘

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

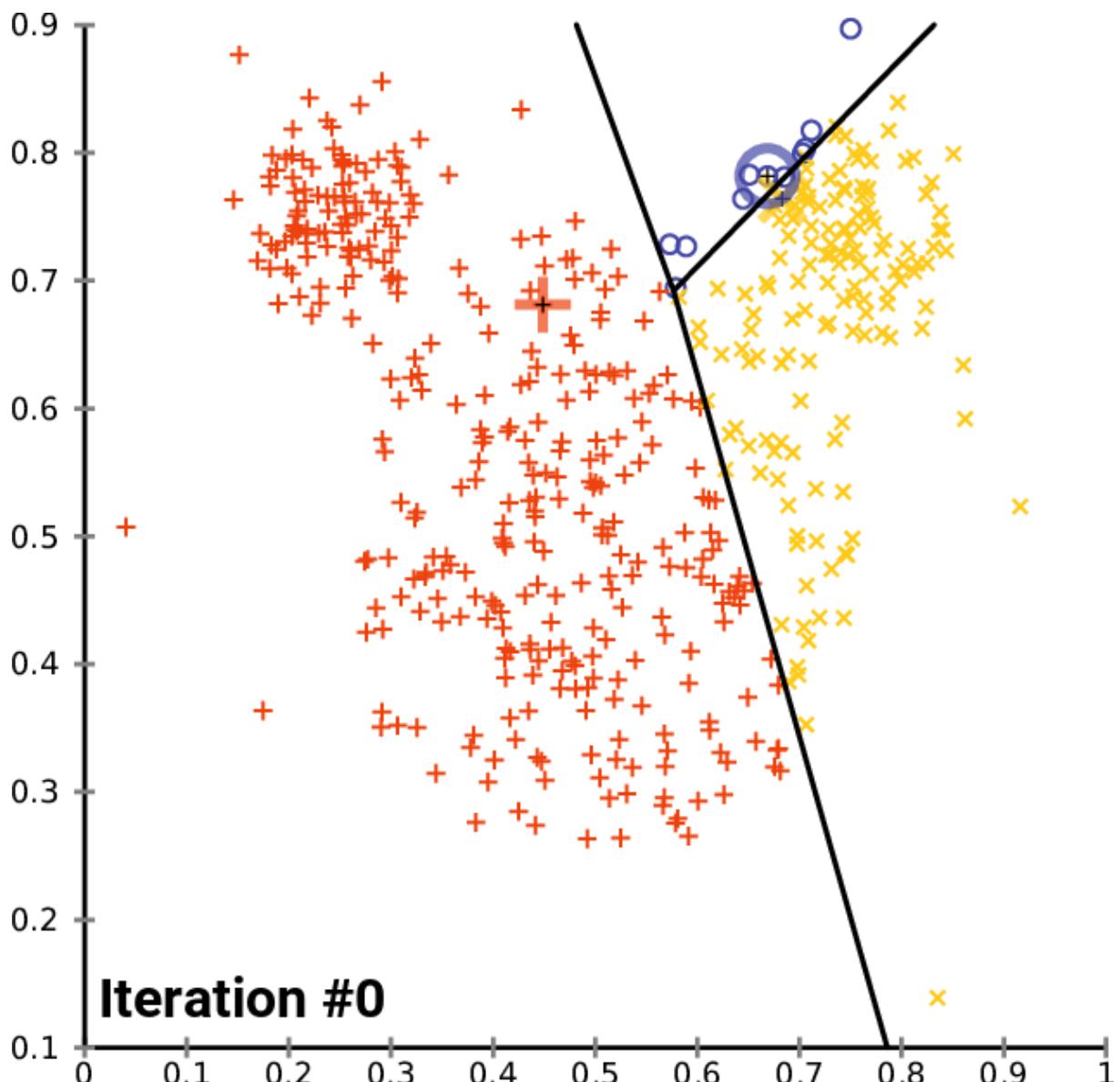
출력: 군집집합 $C = \{c_1, c_2, \dots, c_k\}$

- 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.

1.5 원리



- 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



1.6 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

iris = load_iris()
    
```

Python

```
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

iris = load_iris()
```

✓ 0.9s

Python

1.7 특징 이름 - 항상 뒤에 (cm)가 불편했다

```
iris.feature_names
```

✓ 0.0s

Python

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

1.8 뒷 글자 자르기~

```
cols = [each[:-5] for each in iris.feature_names]
```

✓ 0.0s

Python

```
['sepal length', 'sepal width', 'petal length', 'petal width']
```

1.9 iris 데이터 정리

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

✓ 0.0s

Python

	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

1.10 편의상 두 개의 특성만

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

✓ 0.0s

Python

	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

1.11 군집화 시작~

```
model = KMeans(n_clusters=3)
model.fit(feature)
```

✓ 0.0s

Python

▼ KMeans

```
KMeans(n_clusters=3)
```

```
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

1.12 결과 라벨~ (군집화라서 지도학습의 라벨과 다르다)

```
model.labels_
```

✓ 0.0s

Python

```
array([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, 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, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 2, 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], dtype=int32)
```

1.13 군집 중심값

```
model.cluster_centers_
✓ 0.0s
array([[5.59583333, 2.0375      ],
       [1.462        , 0.246      ],
       [4.26923077, 1.34230769]])
```

Python

1.14 다시 정리 (그림 그리기 위해)

```
predict = pd.DataFrame(model.predict(feature), columns=["cluster"])
feature = pd.concat([feature, predict], axis=1)
feature.head()
✓ 0.0s
```

Python

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

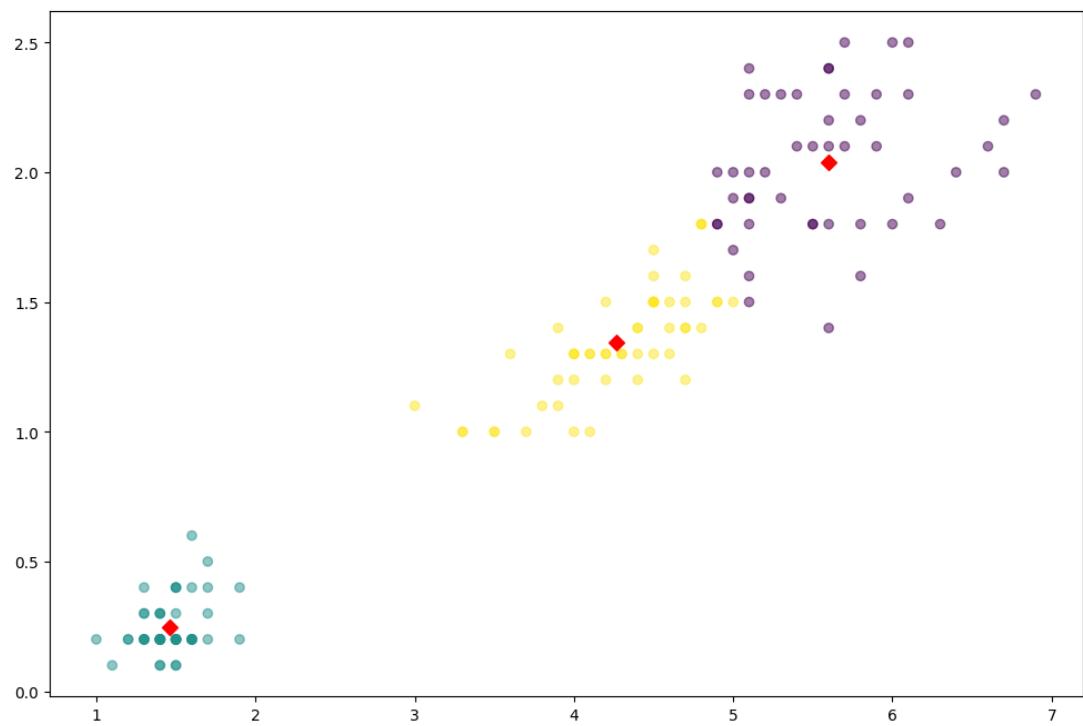
1.15 결과를 확인하기 위해

```
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()
✓ 0.1s
```

Python

1.16 결론



2 make_blobs

2.1 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)
```

0.0s

Python

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

- 군집화 연습을 위한 데이터 생성기

2.2 데이터 정리

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

0.0s

Python

	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

2.3 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
```

0.0s

Python

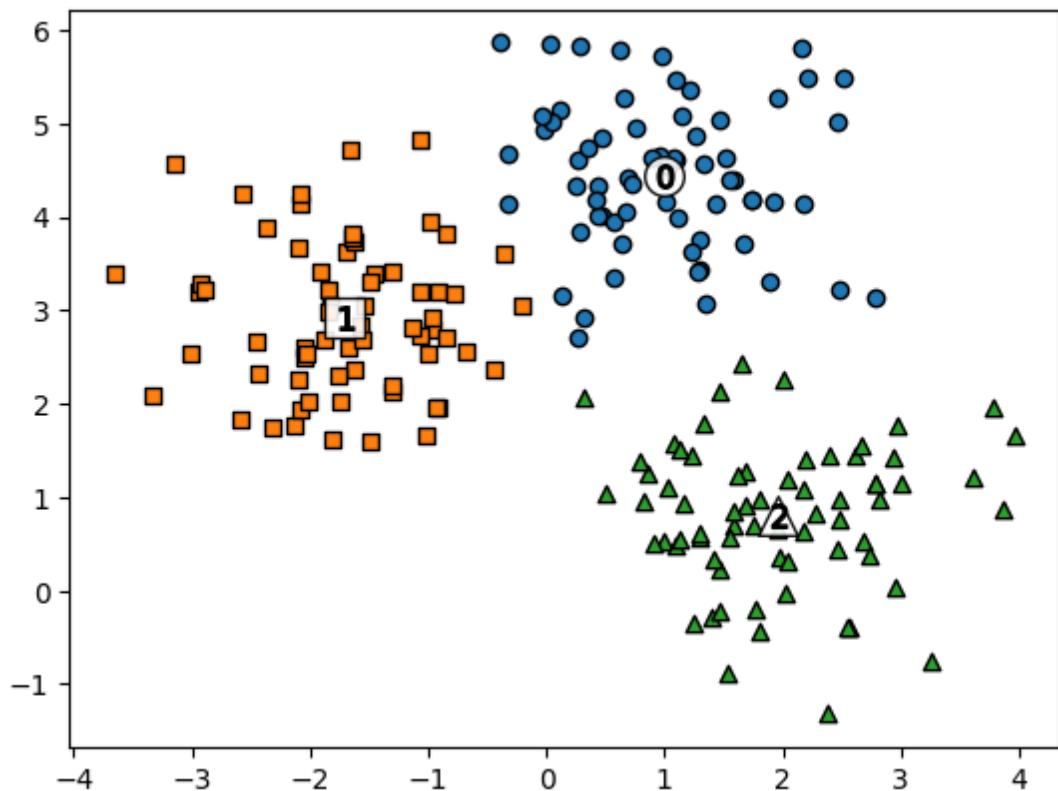
2.4 결과 도식화

```
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,
    )
```

Python

2.5 결론



2.6 결과 확인

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

Python

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

Name: count, dtype: int64

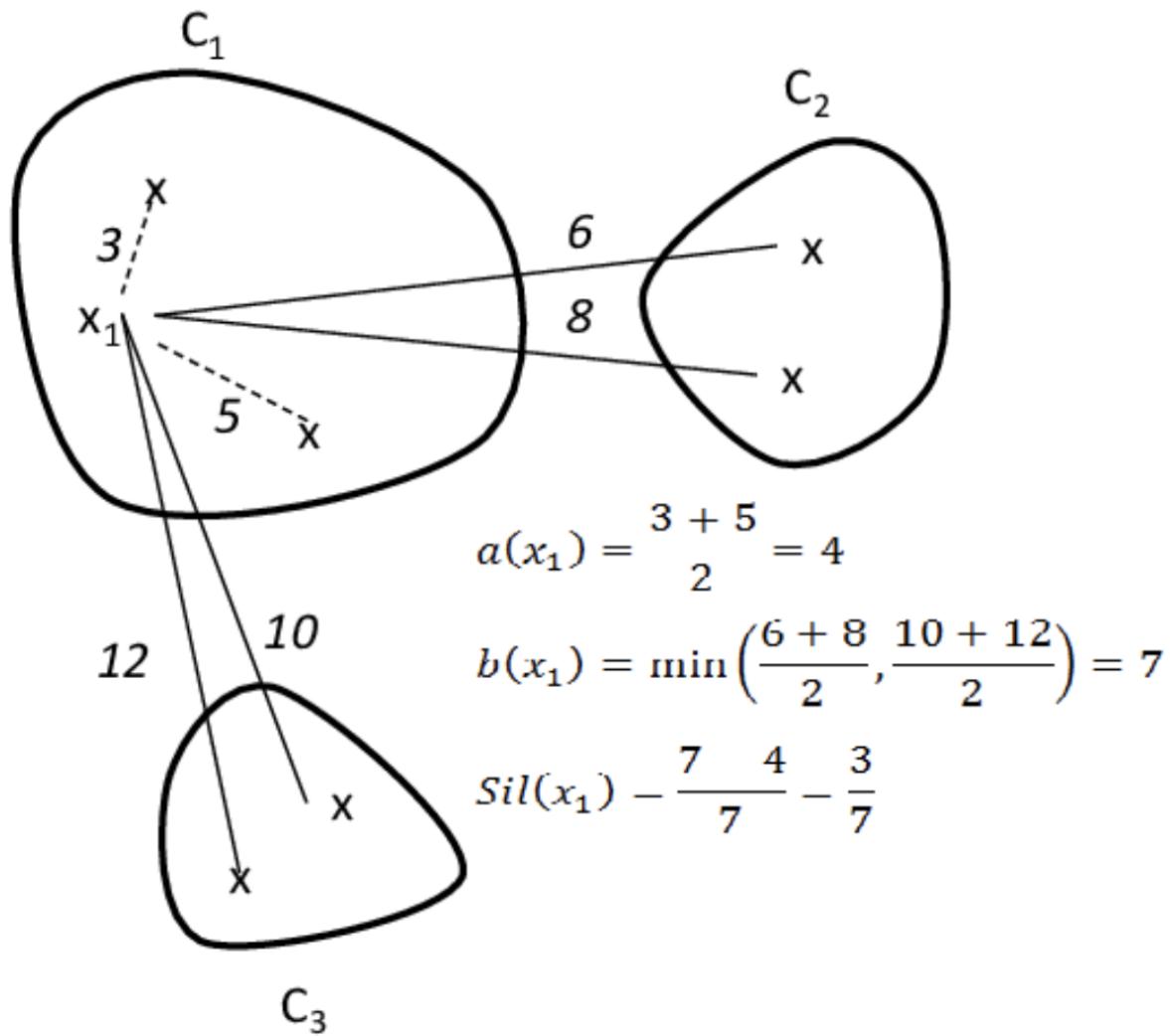
3 군집 평가

3.1 군집 결과의 평가

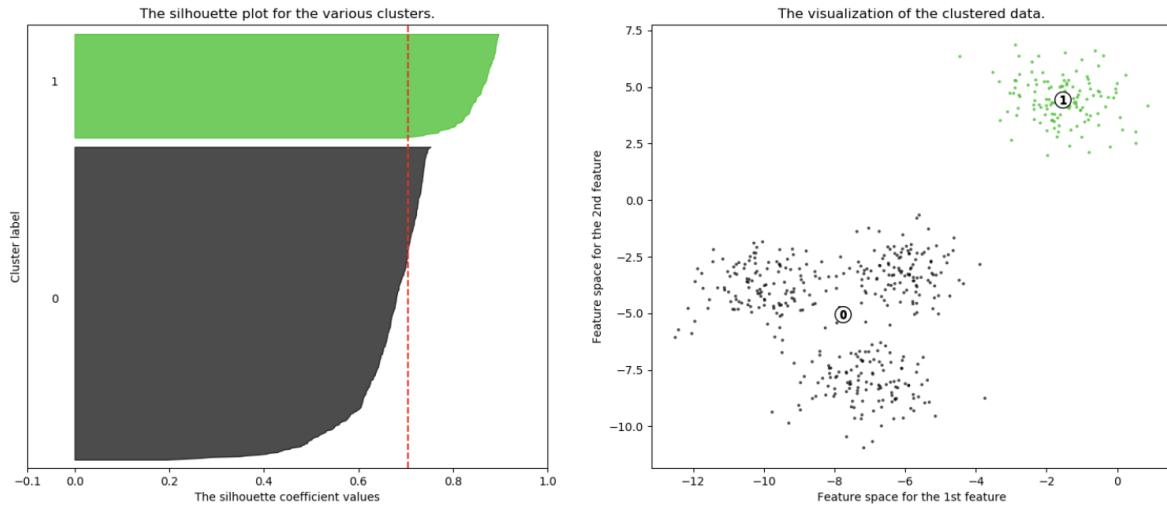
- 분류기는 평가 기준(정답)을 가지고 있지만, 군집은 그렇지 않다.
- 군집 결과를 평가하기 위해 실루엣 분석을 많이 활용한다.

3.2 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

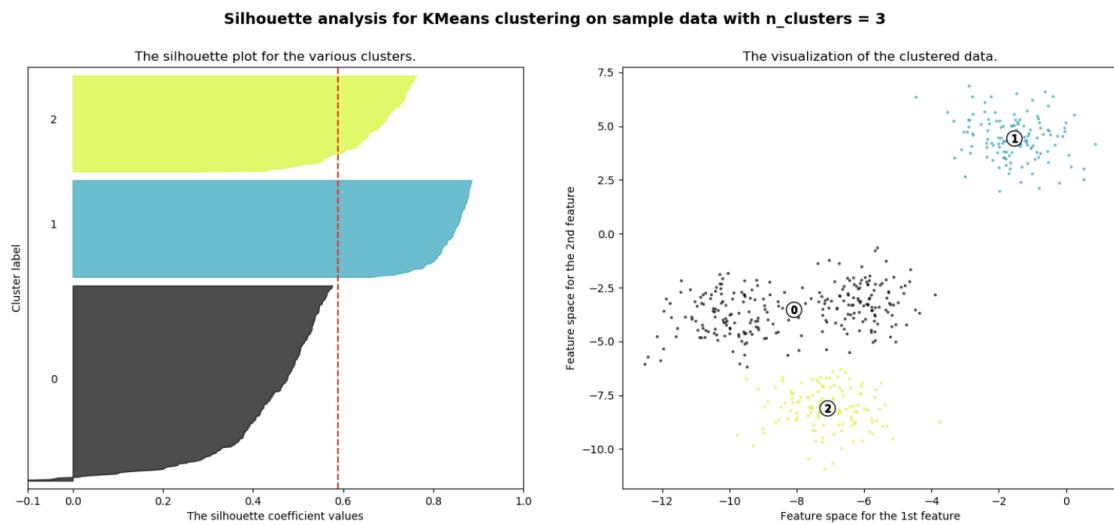


3.3 직접 보면서 이해해 보자 n=2인 경우



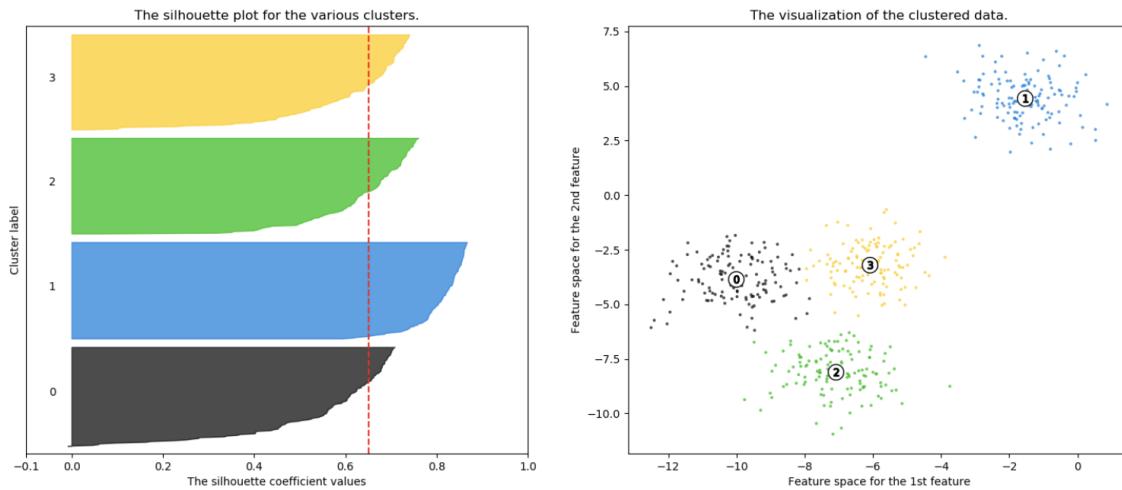
- 1번 군집의 경우 0번 군집과 떨어져 있고 잘 뭉쳐있음
- 0번 군집의 경우 내부 데이터끼리 많이 떨어져 있음

3.4 n=3인 경우



- 0번 군집의 경우 2번 군집과 가깝게 위치해 있음

3.5 n=4인 경우



3.6 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
)
```

Python

3.7 군집 결과 정리하고

```
iris_df[“cluster”] = kmeans.labels_
iris_df.head()
```

Python

	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

3.8 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 0.5528190123564095
silhouette_samples() return 값의 shape  (150,)

```

Python

3.9 실루엣 점수를 시각화

```

def visualize_silhouette(cluster_lists, X_features):

    from sklearn.datasets import make_blobs
    from sklearn.cluster import KMeans
    from sklearn.metrics import silhouette_samples, silhouette_score

    import matplotlib.pyplot as plt
    import matplotlib.cm as cm
    import math

    # 입력값으로 클러스터링 갯수들을 리스트로 받아서, 각 갯수별로 클러스터링을 적용하고 실루엣 개수를 구함
    n_cols = len(cluster_lists)

    # plt.subplots()으로 리스트에 기재된 클러스터링 수만큼의 sub figures를 가지는 axs 생성
    fig, axs = plt.subplots(figsize=(4 * n_cols, 4), nrows=1, ncols=n_cols)

    # 리스트에 기재된 클러스터링 갯수들을 차례로 iteration 수행하면서 실루엣 개수 시각화
    for ind, n_cluster in enumerate(cluster_lists):

        # KMeans 클러스터링 수행하고, 실루엣 스코어와 개별 데이터의 실루엣 값 계산.
        clusterer = KMeans(n_clusters=n_cluster, max_iter=500, random_state=0)
        cluster_labels = clusterer.fit_predict(X_features)

        sil_avg = silhouette_score(X_features, cluster_labels)
        sil_values = silhouette_samples(X_features, cluster_labels)

        y_lower = 10
        axs[ind].set_title(
            "Number of Cluster : " + str(n_cluster) + "\n"
            "Silhouette Score :" + str(round(sil_avg, 3)))
        )
        axs[ind].set_xlabel("The silhouette coefficient values")
        axs[ind].set_ylabel("Cluster label")
        axs[ind].set_xlim([-0.1, 1])
        axs[ind].set_ylim([0, len(X_features) + (n_cluster + 1) * 10])
        axs[ind].set_yticks([])
        # Clear the yaxis labels / ticks
        axs[ind].set_xticks([0, 0.2, 0.4, 0.6, 0.8, 1])

```

```
# 클러스터링 갯수별로 fill_betweenx( )형태의 막대 그래프 표현.
for i in range(n_cluster):
    ith_cluster_sil_values = sil_values[cluster_labels == i]
    ith_cluster_sil_values.sort()

    size_cluster_i = ith_cluster_sil_values.shape[0]
    y_upper = y_lower + size_cluster_i

    color = cm.nipy_spectral(float(i) / n_cluster)
    axs[ind].fill_betweenx(
        np.arange(y_lower, y_upper),
        0,
        ith_cluster_sil_values,
        facecolor=color,
        edgecolor=color,
        alpha=0.7,
    )
    axs[ind].text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))
    y_lower = y_upper + 10

    axs[ind].axvline(x=sil_avg, color="red", linestyle="--")
```

3.10 실루엣 플랏의 결과

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

Python

