강원지역혁신플랫폼

기계학습

Machine Learning

계층적 군집 분석 실습

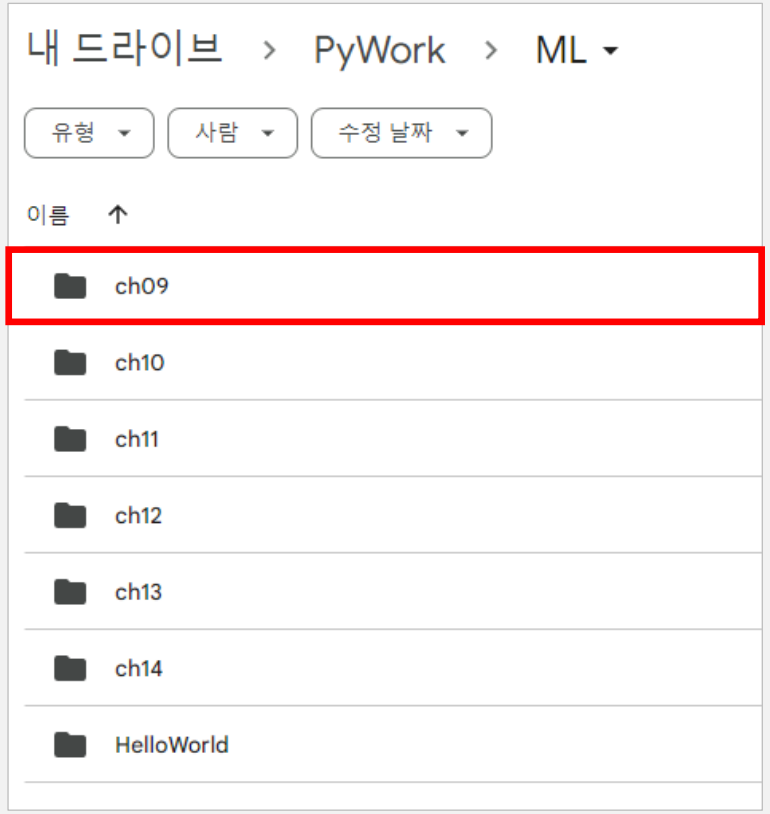
학습목표

- 계층적 군집 분석을 구현할 수 있습니다.

01 | 실습

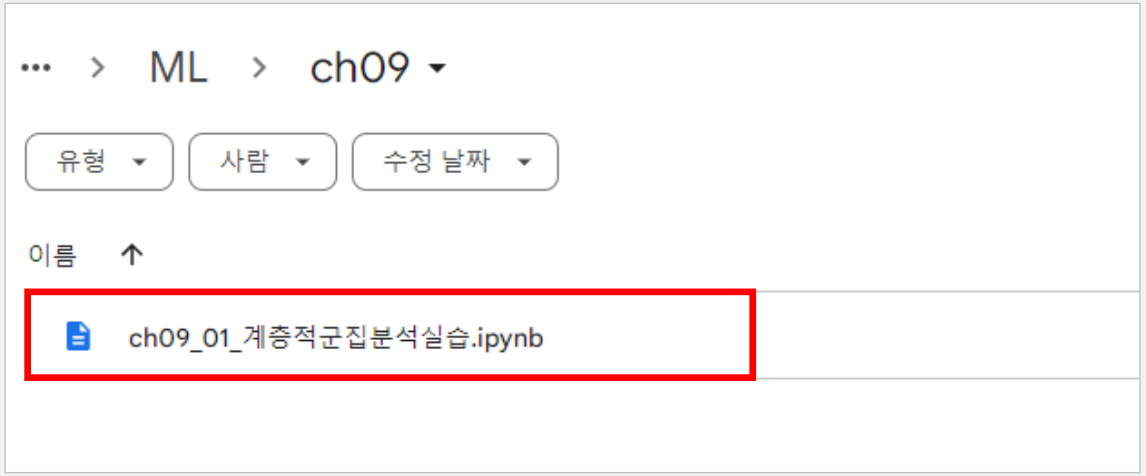
(권장) 아래와 같은 경로에 실행 소스가 존재하면 환경 구축 완료

- 구글 드라이브 "PyWork > ML" 폴더로 이동함
  - 아래의 [ch09] 폴더를 클릭하면 됨



01 | 실습

- “ML > ch09 >” 폴더를 클릭함
  - 아래의 [ch09_01_계층적군집분석실습.ipynb] 스크립트를 클릭함



02 | 계층적 군집 분석 실습 (1)

계층적 군집(Hierarchical clustering) 분석 실습 (1)

- 다음은 임의로 데이터 셋을 생성하여 병합적 계층 군집의 완전 연결방식(Complete linkage method)으로 군집하는 경우를 생각해 보자
  - 아래의 순으로 병합적 계층 군집을 수행함
    - 무작위로 (5, 3) 행렬을 생성함
    - 두 점 사이의 유클리드 거리를 계산함
    - 완전 연결방식으로 병합적 계층 군집을 수행하고, 그 결과를 해석함
    - 병합 계층 군집 결과를 덴드로그램(Dendrogram)으로 그림

02 | 계층적 군집 분석 실습 (1)

- 다음은 무작위로 (5, 3) 행렬의 데이터를 생성하는 코드이다
  - random_sample() 함수는 연속형 균일분포(continuous uniform distribution)를 이용하여 0 이상 1 미만인 범위에서 샘플링된 임의의 실수를 반환함
    - 아래의 경우 0 이상 10 미만인 범위에서 데이터 형상이 (5, 3)으로 데이터를 생성하여 변수 X에 할당함

```
np.random.seed(123)                                  # 시드 설정
X = np.random.random_sample([5, 3]) * 10    # 무작위 함수를 통해 5*3 행렬 생성
print(X.shape)   # (5, 3)
print(X)
#[[6.96469186 2.86139335 2.26851454]
# [5.51314769 7.1946897  4.2310646 ]
# [9.80764198 6.84829739 4.80931901]
# [3.92117518 3.43178016 7.29049707]
# [4.38572245 0.59677897 3.98044255]]
```

02 | 계층적 군집 분석 실습 (1)

- 다음은 무작위로 생성한 (5, 3) 행렬 객체 X를 데이터프레임으로 저장하는 코드이다
  - 실행 결과에서 열 이름은 'x', 'y', 'z' 임
    - 인덱스는 'ID_0' ~ 'ID_4'인 데이터프레임 객체 df에 저장됨

```
variables = ['X', 'Y', 'Z']
labels = ['ID_0', 'ID_1', 'ID_2', 'ID_3', 'ID_4']
df = pd.DataFrame(X, columns = variables, index = labels)
df
```

	X	Y	Z
ID_0	6.964692	2.861393	2.268515
ID_1	5.513148	7.194690	4.231065
ID_2	9.807642	6.848297	4.809319
ID_3	3.921175	3.431780	7.290497
ID_4	4.385722	0.596779	3.980443

02 | 계층적 군집 분석 실습 (1)

- 다음은 무작위로 생성한 (5, 3) 행렬 데이터를 데이터프레임으로 저장한 객체 df를 이용해 유클리드 거리를 계산하는 코드이다
  - scipy.spatial.distance.pdist() 함수는 주어진 점들 사이의 모든 거리를 계산함
    - 실행결과 아래와 같이 두 점 사이의 유클리드 거리 계산 결과 10개의 값이 존재하는 것을 볼 수 있음

```
distmatrix = pdist(df, metric='euclidean')  # 두 점 사이의 유클리드 거리 계산
print(distmatrix.shape)                     # (10,)
print(distmatrix)
```

```
(10,)
[4.973534   5.51665266 5.89988504 3.83539555 4.34707339 5.10431109
 6.69823298 7.24426159 8.31659367 4.382864  ]
```

02 | 계층적 군집 분석 실습 (1)

다음은 무작위로 생성한 (5, 3) 행렬 데이터를 데이터프레임으로 저장한 객체 df를 이용해 유클리드 거리를 구하고, 거리행렬 벡터를 행렬 형식으로 변환하여 반환하는 코드이다

- scipy.spatial.distance.squareform() 함수는 거리 행렬 벡터를 행렬 형식으로 변환하여 반환함

```
row_dist = pd.DataFrame(squareform(pdist(df, metric='euclidean')), columns=labels, index=labels)
row_dist
print(row_dist.shape)  # (5, 5)
print(row_dist)
#              ID_0      ID_1      ID_2      ID_3      ID_4
# ID_0 0.000000 4.973534  5.516653  5.899885  3.835396
# ID_1 4.973534  0.000000  4.347073  5.104311  6.698233
# ID_2 5.516653  4.347073  0.000000  7.244262  8.316594
# ID_3 5.899885  5.104311  7.244262  0.000000  4.382864
# ID_4 3.835396  6.698233  8.316594  4.382864  0.000000
```

02 | 계층적 군집 분석 실습 (1)

다음은 병합적 방법인 **완전 연결 방식**으로 **계층적 군집**을 수행하는 코드이다

- 아래의 결과에서 **군집ID_1**, **군집ID_2**는 **각 군집**에서 **완전 연결 방식**으로 **병합된 군집**을 나타냄
  - 거리(distance)는 군집 간의 거리임
  - 군집화 결과는 **무작위 생성 데이터 5개**를 **4개의 군집**으로 **군집화**한 것임

```
row_clusters = linkage(df.values, metric='euclidean', method='complete')
pd.DataFrame(row_clusters, columns=['군집ID_1', '군집ID_2', '거리', '군집 멤버수'],
                           index=['군집 %d' % (i+1) for i in range(row_clusters.shape[0])])
```

	군집ID_1	군집ID_2	거리	군집 멤버수
군집 1	0.0	4.0	3.835396	2.0
군집 2	1.0	2.0	4.347073	2.0
군집 3	3.0	5.0	5.899885	3.0
군집 4	6.0	7.0	8.316594	5.0

군집 결과

	ID_0	ID_1	ID_2	ID_3	ID_4
ID_0	0.000000	4.973534	5.516653	5.899885	3.835396
ID_1	4.973534	0.000000	4.347073	5.104311	6.698233
ID_2	5.516653	4.347073	0.000000	7.244262	8.316594
ID_3	5.899885	5.104311	7.244262	0.000000	4.382864
ID_4	3.835396	6.698233	8.316594	4.382864	0.000000

거리 행렬 벡터

02 | 계층적 군집 분석 실습 (1)

앞의 계층적 군집 결과를 해석해 보자

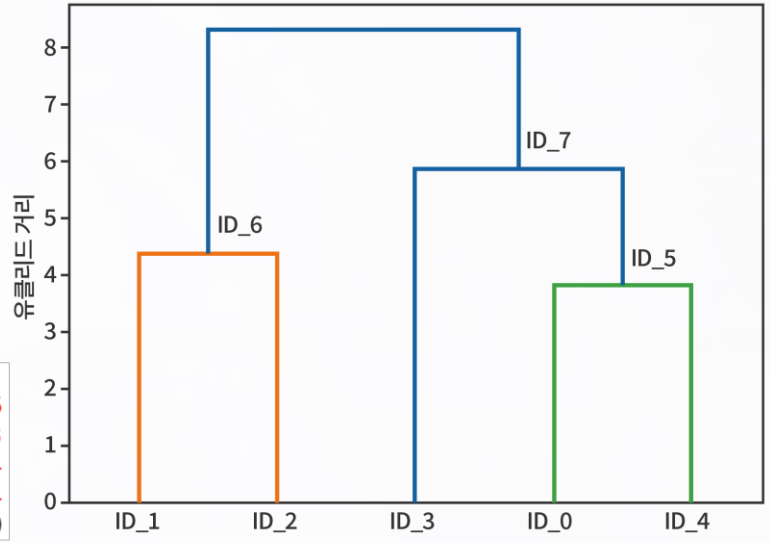
- 군집1의 경우, ID_0과 ID_4가 병합됨 → 이게 ID_5가 되는 것임
- 군집2의 경우, ID_1과 ID_2가 병합됨 → 이게 ID_6이 되는 것임
- 군집3의 경우, ID_3과 ID_5(ID_0 + ID_4)가 병합됨 → ID_7이 되는 것임
- 군집4의 경우, ID_6(ID_1 + ID_2)과 ID_7(ID_3 + (ID_0 + ID_4))가 병합된 것임

	군집ID_1	군집ID_2	거리	군집 멤버수
군집 1	0.0	4.0	3.835396	2.0
군집 2	1.0	2.0	4.347073	2.0
군집 3	3.0	5.0	5.899885	3.0
군집 4	6.0	7.0	8.316594	5.0

군집 결과

	ID_0	ID_1	ID_2	ID_3	ID_4
ID_0	0.000000	4.973534	5.516653	5.899885	3.835396
ID_1	4.973534	0.000000	4.347073	5.104311	6.698233
ID_2	5.516653	4.347073	0.000000	7.244262	8.316594
ID_3	5.899885	5.104311	7.244262	0.000000	4.382864
ID_4	3.835396	6.698233	8.316594	4.382864	0.000000

거리 행렬 벡터



덴드로그램

02 | 계층적 군집 분석 실습 (1)

✦ 군집 멤버수는 군집에 속한 데이터의 수임

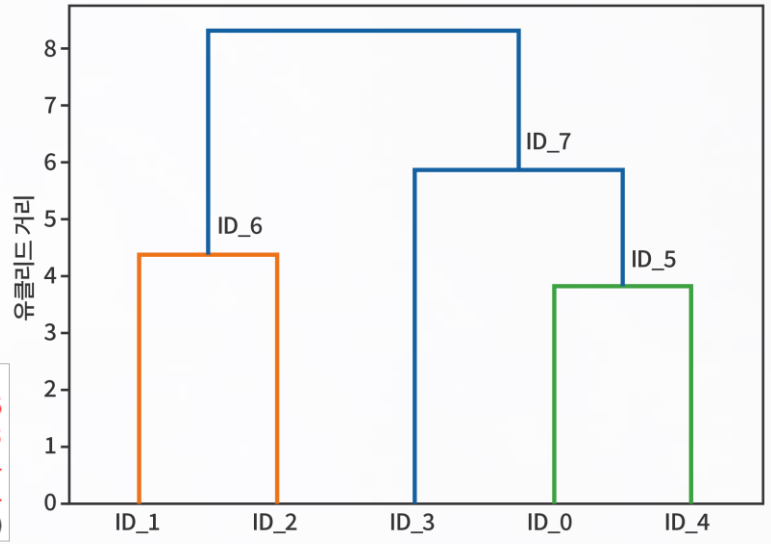
- 군집1 : ID_0, ID_4 → ID_5(ID_0, ID_4) 2개
- 군집2 : ID_1, ID_2 → ID_6(ID_1, ID_2) 2개
- 군집3 : ID_5(ID_0, ID_4), ID_3 → ID_7(ID_0, ID_4, ID_3) 3개
- 군집4 : ID_6(ID_1, ID_2), ID_7(ID_3, ID_0, ID_4)가 병합된 것임 5개

	군집ID_1	군집ID_2	거리	군집 멤버수
군집 1	0.0	4.0	3.835396	2.0
군집 2	1.0	2.0	4.347073	2.0
군집 3	3.0	5.0	5.899885	3.0
군집 4	6.0	7.0	8.316594	5.0

군집 결과

	ID_0	ID_1	ID_2	ID_3	ID_4
ID_0	0.000000	4.973534	5.516653	5.899885	3.835396
ID_1	4.973534	0.000000	4.347073	5.104311	6.698233
ID_2	5.516653	4.347073	0.000000	7.244262	8.316594
ID_3	5.899885	5.104311	7.244262	0.000000	4.382864
ID_4	3.835396	6.698233	8.316594	4.382864	0.000000

거리 행렬 벡터



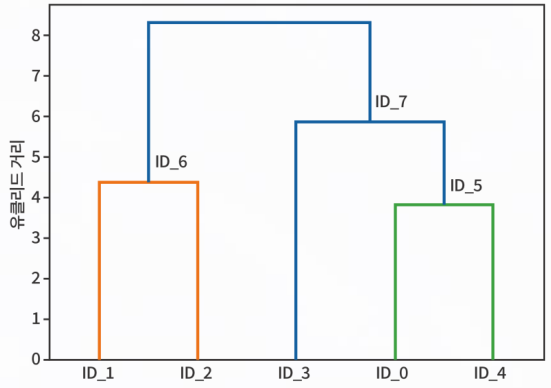
덴드로그램

02 | 계층적 군집 분석 실습 (1)

다음은 병합적 방법인 **완전 연결 방식**으로 **계층적 군집** 결과를 **덴드로그램**으로 그리는 코드이다

- 계층 군집 결과로 덴드로그램(Dendrogram)을 그릴 수 있음
  - 이는, **의미 있는 분류 체계**를 **만들어**줌

```
row_dendrogram = dendrogram(row_clusters, labels=labels)
plt.rcParams["font.family"] = 'Malgun Gothic'
plt.tight_layout()
plt.ylabel('유클리드 거리')
plt.show()
```



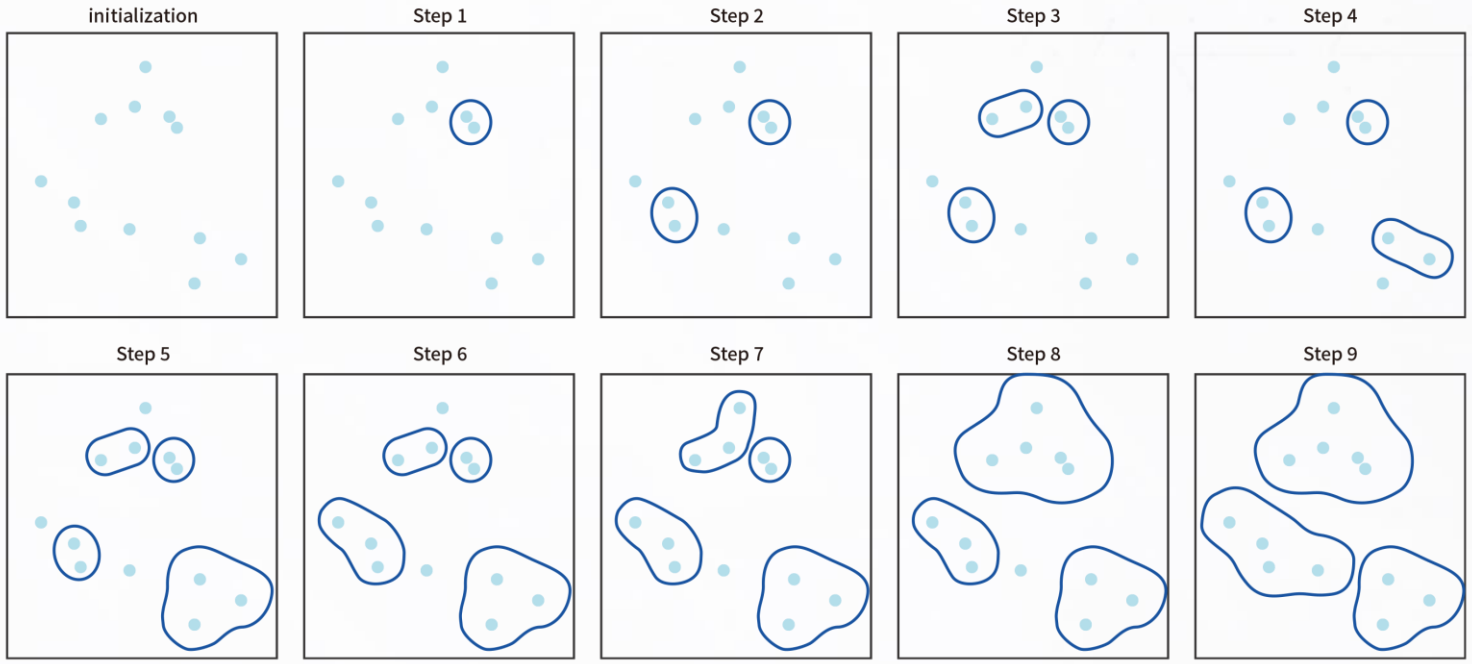
덴드로그램

	군집ID_1	군집ID_2	거리	군집 멤버수
군집 1	0.0	4.0	3.835396	2.0
군집 2	1.0	2.0	4.347073	2.0
군집 3	3.0	5.0	5.899885	3.0
군집 4	6.0	7.0	8.316594	5.0

군집 결과

계층적 군집 분석 실습 (2)

- 다음은 2차원 데이터 세트에서 세 개의 군집을 찾기 위한 계층적 군집의 과정임
  - 여기서는 병합적 방법으로 계층적 군집을 수행함

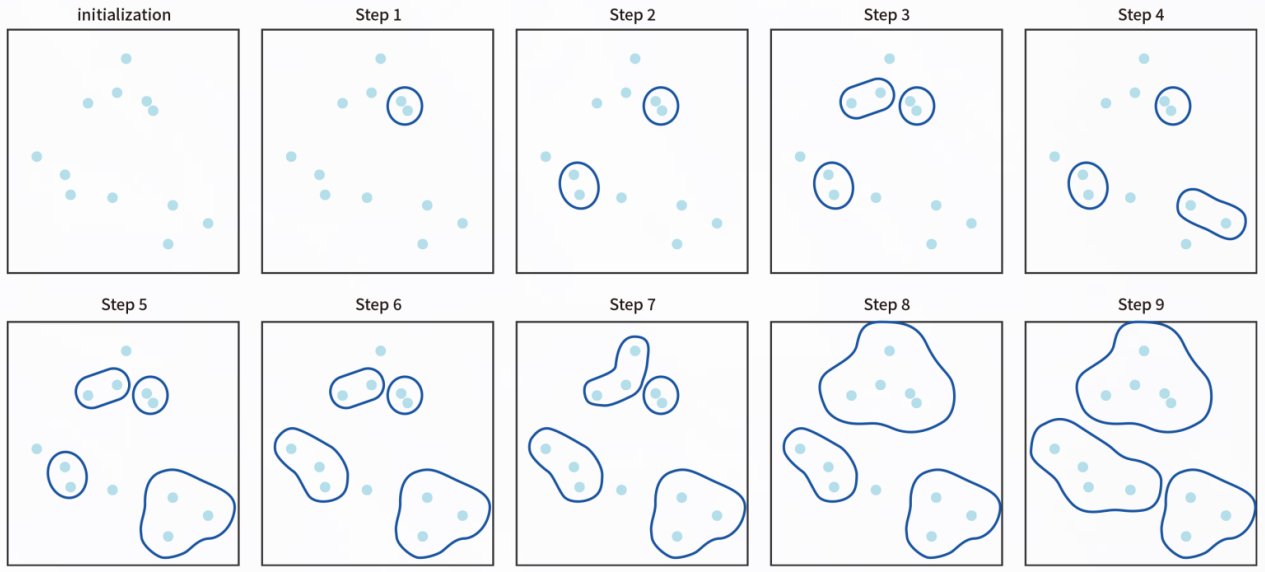


03 | 계층적 군집 분석 실습 (2)

다음은 병합적 방법으로 계층적 군집 과정을 시각화하는 코드이다.

- 아래 그림과 같이 병합적 방법은 Bottom-Up 접근 방식인 것을 볼 수 있음

```
mglearn.plots.plot_agglomerative_algorithm()    # 알고리즘 설명 시각화
plt.show()
```



03 | 계층적 군집 분석 실습 (2)

아래의 그림은 2차원 데이터셋에서 세 개의 군집을 찾기 위한 병합적 군집의 과정임

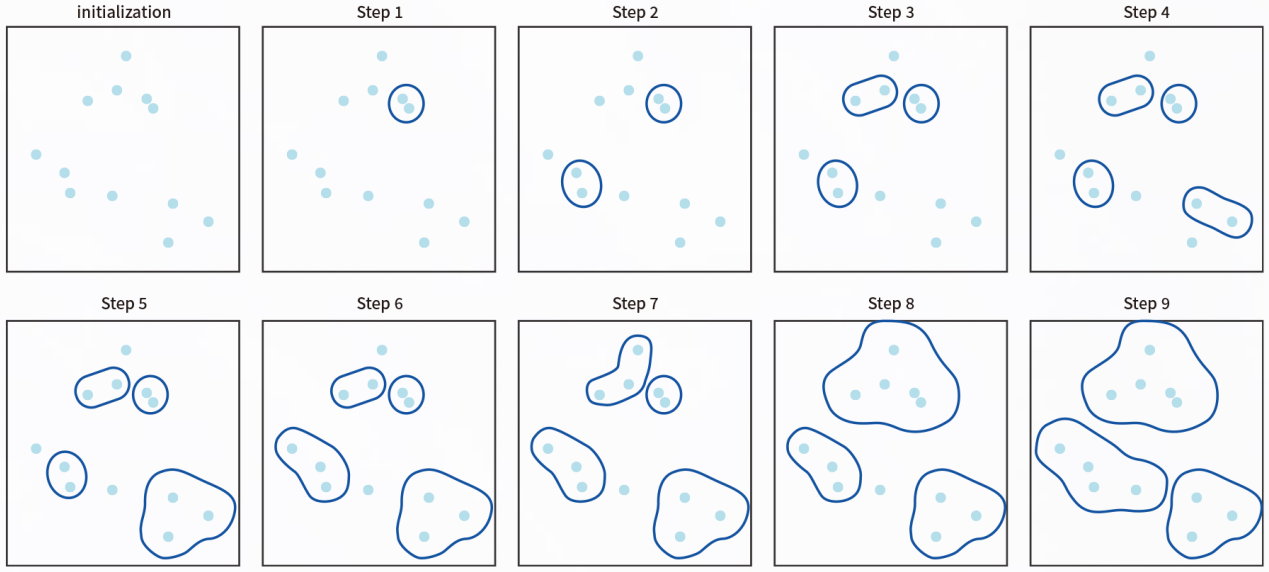
✦ 병합적 방법의 계층적 군집 알고리즘 :

› 시작할 때 각 포인트를 하나의 군집으로 지정함

› 특정 종료 조건을 만족할 때까지 가장 비슷한 두 군집을 합침

› 종료조건

- 군집 개수(아래의 경우 3개의 군집), 지정된 개수의 군집이 남을 때까지 비슷한 군집을 합침

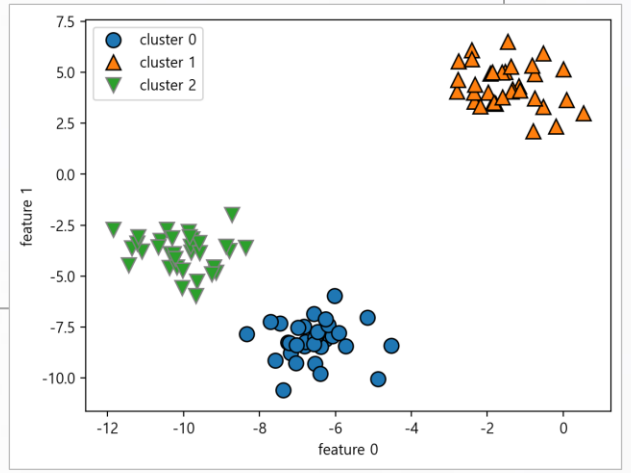


03 | 계층적 군집 분석 실습 (2)

다음은 아래의 조건을 만족하는 코드이다.

- 임의로 생성된 데이터셋은 2개의 독립변수, 3개의 군집으로 구성됨
  - 이 데이터로 계층적 군집을 수행하여 예측 정보(소속 정보)을 얻어 산점도를 그림

```
X, y = make_blobs(random_state=1)
print(X.shape, y.shape)      # (100, 2) (100,)

# 완전연결방식, n_clusters=3
agg = AgglomerativeClustering(n_clusters=3, affinity='euclidean', linkage='complete')
assignment = agg.fit_predict(X)
mglearn.discrete_scatter(X[:, 0], X[:, 1], assignment)
plt.legend(["cluster 0", "cluster 1", "cluster 2"], loc="best")
plt.xlabel("feature 0")
plt.ylabel("feature 1")
```



03 | 계층적 군집 분석 실습 (2)

- 병합적 방법의 계층적 군집 알고리즘의 특성상 새로운 데이터 포인트에 대해서는 예측을 할 수 없음
  - 그러므로, 병합적 방법의 계층적 군집은 predict() 메서드가 없음
    - 그래서, 군집을 만들고 소속 정보를 얻기 위해 fit_predict() 메서드를 사용함

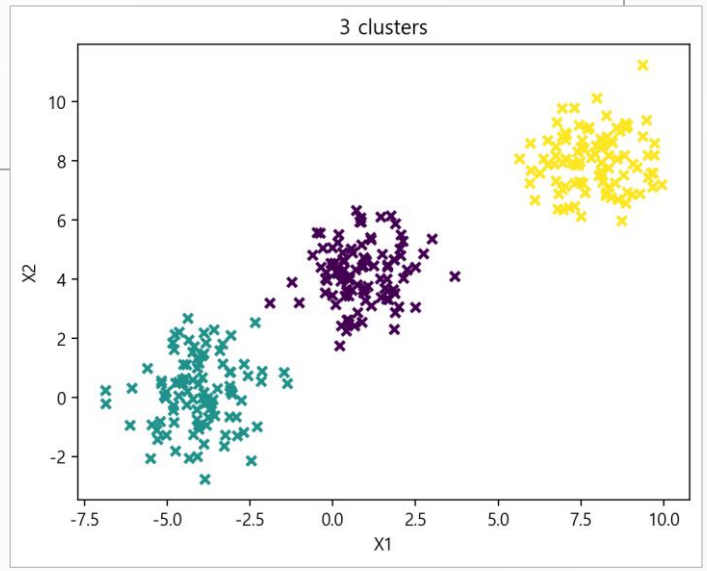
```
X, y = make_blobs(random_state=1)
print(X.shape, y.shape)      # (100, 2) (100,)

# 완전연결방식, n_clusters=3
agg = AgglomerativeClustering(n_clusters=3, affinity='euclidean', linkage='complete')
assignment = agg.fit_predict(X)
mglearn.discrete_scatter(X[:, 0], X[:, 1], assignment)
plt.legend(["cluster 0", "cluster 1", "cluster 2"], loc="best")
plt.xlabel("feature 0")
plt.ylabel("feature 1")
```

03 | 계층적 군집 분석 실습 (2)

다음은 아래의 조건을 만족하는 코드이다.

- 임의로 생성된 데이터셋은 2개의 독립변수, 3개의 군집, 300개의 표본으로 구성됨
  - 이 데이터로 산점도를 그림

```
X, y = make_blobs(n_samples=300, n_features=2, centers=3, random_state=3)
plt.title("3 clusters")
plt.scatter(X[:, 0], X[:, 1], marker='x', c=y, s=30, edgecolor="k", linewidth=2)
plt.xlabel("X1")
plt.ylabel("X2")
plt.show()
```



계층적 군집 분석 실습 (3)

- 다음은 아이리스(iris) 데이터셋으로 병합적 방법의 계층적 군집을 수행해 보자
  - 아이리스 데이터 셋은 꽃잎의 각 부분의 너비와 길이 등을 측정한 데이터임
    - 관측치는 150개, 속성은 6개로 구성되어 있음
    - 아이리스 꽃은 아래 그림과 같음

열이름	설명
Caseno	일련번호
Sepal Length	꽃받침의 길이 정보
Sepal Width	꽃받침의 너비 정보
Petal Length	꽃잎의 길이 정보
Petal Width	꽃잎의 너비 정보
Species	꽃의 종류 정보 (setosa, versicolor,virginica)

04 | 계층적 군집 분석 실습 (3)

다음은 아이리스(iris) 데이터셋을 읽어서 데이터 프레임 형식으로 저장하는 코드이다.

- 독립 변수: ‘Sepal_length’, ‘Sepal_width’, ‘Petal_length’, ‘Petal_width’
- 종속 변수: labels

```
Iris = load_iris()
df = pd.DataFrame(iris.data, columns=['Sepal_length','Sepal_width','Petal_length','Petal_width'])
df['labels'] = iris.target
df
```

	Sepal_length	Sepal_width	Petal_length	Petal_width	labels
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

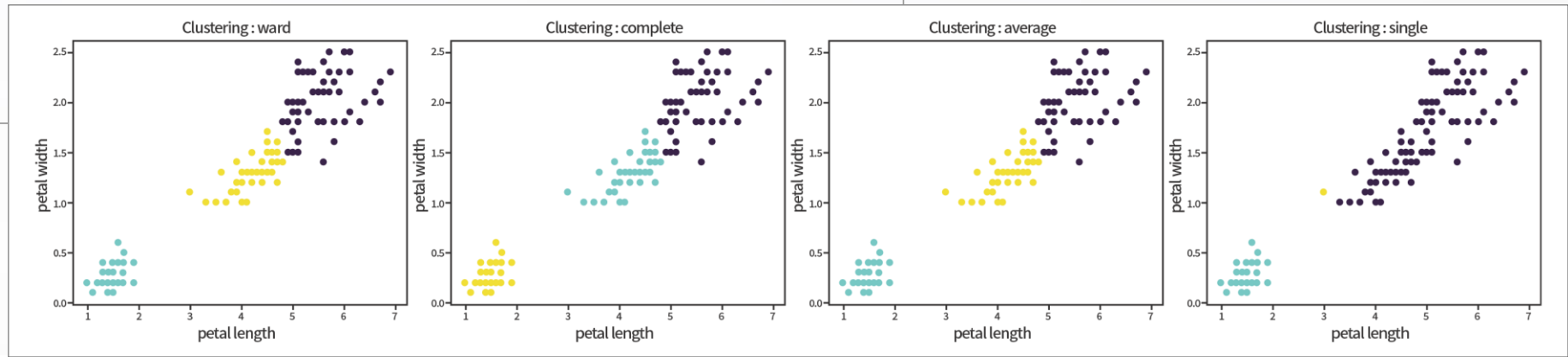
150 rows × 5 columns

04 | 계층적 군집 분석 실습 (3)

다음은 아이리스 데이터셋으로 Ward, 완전, 평균, 단일 연결법으로 병합적 방법의 계층적 군집을 수행 후 산점도를 그리는 코드이다.

✦ 여기서는 아이리스 데이터셋의 “Petal_length”, “Petal_width” 두 개의 속성을 사용

```
linkage = ['ward', 'complete', 'average', 'single']
for idx, i in enumerate(linkage):
    plt.figure(idx)
    hier = AgglomerativeClustering(n_clusters=3, affinity='euclidean', linkage=i)
    hier.fit(df.iloc[:, 2:4])                    # 'petal length'와 'petal width' column을 사용해 학습
    plt.scatter(df.iloc[:, 2], df.iloc[:, 3], c=hier.labels_)
    plt.title('Clustering : ' + i)
    plt.xlabel('petal length')
    plt.ylabel('petal width')
plt.show()
```

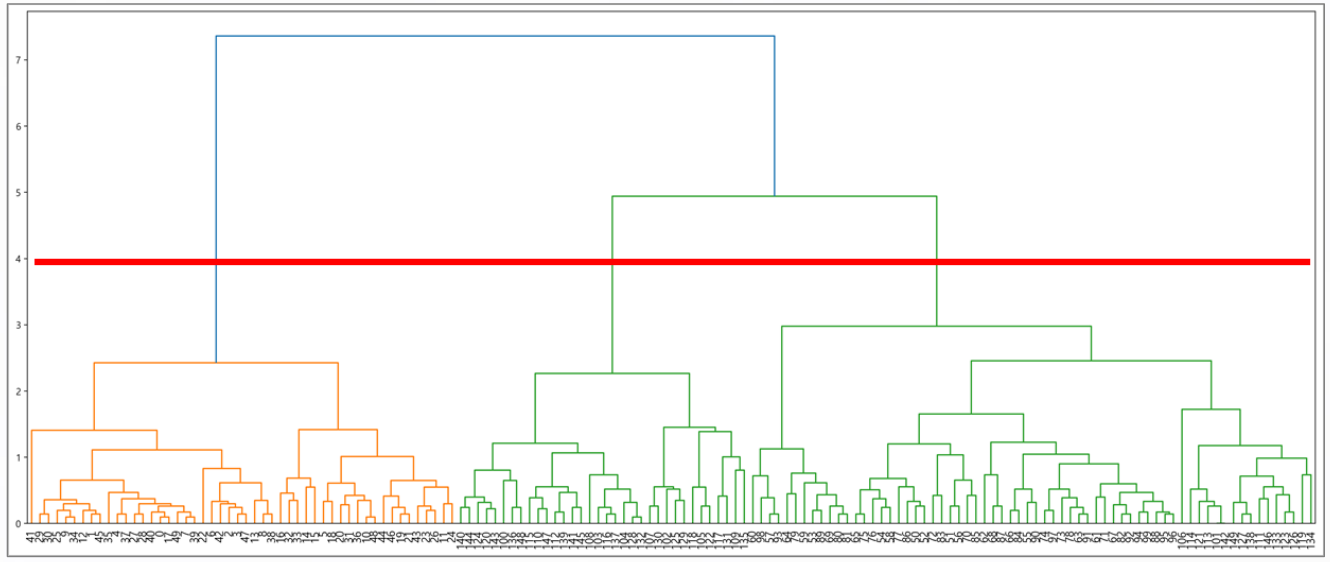


04 | 계층적 군집 분석 실습 (3)

다음은 아이리스 데이터 셋으로 **완전 연결법**으로 **계층적 군집**을 **수행**하고, 그 **결과**로 **덴드로그램**을 그리는 코드이다.

- 아래 그림과 같이 **빨간색 선**으로 **3개 군집**으로 나눌 수 있음

```
clustering = linkage(df, method='complete')        # 완전 연결방식
plt.figure(figsize=(25,10))
dendrogram(clustering, leaf_rotation=90, leaf_font_size=12)
plt.show()
```



04 | 계층적 군집 분석 실습 (3)

다음은 아이리스 데이터 셋으로 완전 연결법의 계층적 군집을 수행하고, 그 결과로 지정한 군집을 자르는 코드이다.

- fcuster() 함수는 특정 y값에서 군집을 자를 수 있음
  - 여기서는 y의 임계값을 3으로 지정하여 1, 2, 3으로 입력 데이터를 반환함
    - 아래 결과에서 2와 3이 섞여 있음을 볼 수 있음

```
mergings = linkage(df, method='complete')  # 완전 연결방식
y_predict = fcluster(mergings, t=3, criterion='distance')
y_predict
```

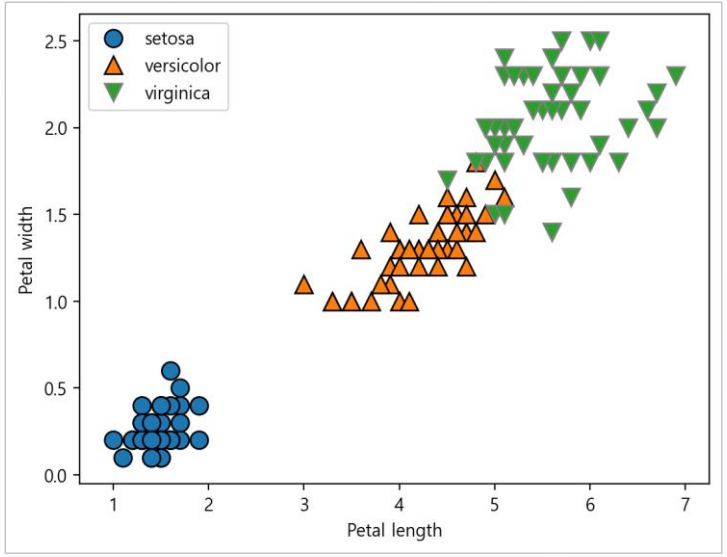
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 3, 2, 2, 2, 2, 3, 2, 2, 2,
       2, 3, 2, 3, 3, 2, 2, 2, 2, 3, 2, 3, 2, 3, 2, 2, 3, 3, 2, 2, 2, 2,
       2, 3, 3, 2, 2, 2, 3, 2, 2, 2, 3, 2, 2, 2, 3, 2, 2, 3], dtype=int32)
```

04 | 계층적 군집 분석 실습 (3)

다음은 아이리스 데이터셋으로 산점도를 그리는 코드이다.

- 실행결과 versicolor와 virginica의 일부 데이터가 섞여 있는 것을 볼 수 있음

```
mglearn.discrete_scatter(df.iloc[:, 2], df.iloc[:, 3], df.iloc[:, 4])
plt.legend(["setosa", "versicolor", "virginica"], loc="best")
plt.xlabel("Petal length")
plt.ylabel("Petal width")
```



04 | 계층적 군집 분석 실습 (3)

다음은 완전 연결법의 계층적 군집을 수행하고, 아이리스의 “Petal_length” 와 “Petal_width” 속성으로 소속정보를 얻어 산점도를 그리는 코드이다.

아래 결과와 같이 잘 군집된 것을 볼 수 있음

```
agg = AgglomerativeClustering(n_clusters=3, affinity='euclidean', linkage='complete')
assignment = agg.fit_predict(df.iloc[:, 2:4])  # 'petal length'와 'petal width' column을 사용해 학습

mglearn.discrete_scatter(df.iloc[:, 2], df.iloc[:, 3], assignment)
plt.legend(["setosa", "versicolor", "virginica"], loc="best")
plt.xlabel("Petal length")
plt.ylabel("Petal width")
```