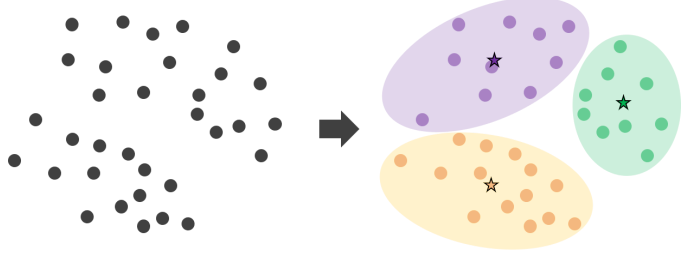
5주차 _ 10조

K-평균 군집화 모델의 평가지표인 Rand지수와 실루엣 계수에 대해 설명해보시오.

📌 K-평균 군집화

비지도 학습 알고리즘 중 하나로, 주어진 데이터를 K개의 군집으로 나누는 작업을 수행한다. 각 군집은 중심이라 불리는 점으로 나타내며, 데이터 포인트는 가장 가까운 중심에 할당된다. **과정**



✅
알고리즘 동작 과정
1.
중심 초기화:

- K개의 군집 중심을 초기화. 무작위로 선택하거나 특정 방법으로 초기화할 수 있다.

2. **할당 단계 :**
    - 각 데이터 포인트를 가장 가까운 중심에 할당. 이는 유클리드 거리 또는 다른 거리 측정 기준을 사용하여 계산된다.
3. **업데이트 단계:**
    - 각 군집의 중심을 해당 군집에 속한 데이터 포인트들의 평균으로 업데이트한다.
4. **할당과 업데이트 반복:**
    - 할당과 업데이트 단계를 반복하면서 중심과 군집 할당이 수렴할 때까지 진행한다.

알고리즘이 수렴하면, 각 데이터 포인트는 K개의 군집 중 하나에 속하게 되며, 각 군집은 그에 해당하는 중심을 가지게 된다.

✏️ Rand 지수 (Rad Index)

- **Rand 지수는 군집화 결과의 정확도를 측정하는 지표** 중 하나
- 공식은 아래와 같다.

$$R = \frac{a+b}{\binom{n}{2}}$$

두 군집 간의 유사성을 측정하는데 사용되는 지수이며식에서의 각 값은 다음을 나타낸다.
-
a : 2개의 방법에서 같은 군집에 있는 쌍
-
b : 2개의 방법에서 서로 다른 군집에 있는 쌍
-
$\binom{n}{2}$: 전체 경우의 수

예를 들어 간단하게
$n = 5$인 데이터셋 $\{\mathrm{A},\mathrm{B},\mathrm{C},\mathrm{D},\mathrm{E}\}$가 있다고 할 때 밑과 같이 2개의 군집으로 나눠볼 수 있다.
- 방법 1 :

$$\begin{cases} \text{cluster 1} : \{\mathrm{A},\mathrm{B}\}, \\ \text{cluster 2} : \{\mathrm{C},\mathrm{D},\mathrm{E}\} \end{cases}$$

- 방법 2 :

$$\begin{cases} \text{cluster 1} : \{\text{A}\}, \\ \text{cluster 2} : \{\text{B,C}\}, \\ \text{cluster 3} : \{\text{D,E}\} \end{cases}$$

전체 경우의 수는
$\binom{5}{2} = 10$이고 $\{\text{A,B}\}, \{\text{A,C}\}, \{\text{A,D}\}, \{\text{A,E}\}, \{\text{B,C}\}, \{\text{B,D}\}, \{\text{B,E}\}, \{\text{C,D}\}, \{\text{C,E}\}, \{\text{D,E}\}$ 이다.

이 때 전체 경우 중
$\{\text{D,E}\}$ 만이 2개의 방법에서 모두 cluster 2에 속하므로 $a = 1$이고, , $\{\text{A,C}\}, \{\text{A,D}\}, \{\text{A,E}\}, \{\text{B,D}\}, \{\text{B,E}\}$ 5개 경우가 2개 방법에서 항상 서로 다른 군집에 속하므로 $b = 5$이다.
따라서
$\text{Rand index} = \dfrac{1+5}{10} = 0.6$이다.

- 그러나 군집의 개수가 많아지면 두 데이터가 각 방법에서 서로 다른 군집에 속할 확률 즉 b값이 자연스레 커지게 되므로 Rand index도 커지게 되므로 수정된 Rand index_ 를 사용하여 군집의 유사성을 측정하게 된다.
- **범위는 0~1, 1에 가까울수록 좋은 성능**

✏️ 수정 Rand 지수 (adjusted Rand index, ARI)

- **Rand Index의 문제점인 유사도를 과대평가하는 특성을 완화하기 위해 사용**된다.
- 특히, KMeans와 같은 군집화 알고리즘의 성능을 평가하는 데 활용된다.
- 공식은 다음과 같다

  $ARI = \frac{RI - E[RI]}{max(RI) - E[RI]}$
  여기서,
  - RI: Rand Index
  - ($E[RI]$: 예상되는 Rand Index (랜덤 군집화 결과에 대한 평균)
- 1과 1 사이의 값을 가지며, **1에 가까울수록 높은 군집화 성능**을 나타낸다.
- 실제 레이블이 없는 경우에는 사용할 수 없으며, 실제값을 데이터화한 경우가 적다는 한계가 있다.

✏️ 실루엣 계수 (Silhouette Coefficient)

- **실루엣 계수는 군집 내의 데이터가 얼마나 조밀하게 모여 있고, 군집 간의 거리가 얼마나 떨어져 있는지**를 나타내는 지표
- 각 데이터 포인트에 대해 실루엣 값이 계산되고, 이를 평균한 값이 모델의 실루엣 계수가 된다.
- **실루엣 값은 -1 ~ 1까지의 범위를 가지며, 높을수록 군집화의 품질이 좋다**고 판단한다.
- 이를 정의하기 위한 수식은 다음과 같다.

  군집 내 거리 $a(i)$: i번째 데이터가 속한 C_I 군집 내에서 자기 자신(i번째 데이터)과 군집 내 다른 모든 데이터 (j번째 데이터) 와의 평균 거리다. 이는 군집 내의 데이터가 얼마나 가깝게 모여 있는지를 나타낸다.

  $$a(i) = \frac{1}{|C_I| - 1} \sum_{j \in C_I, j \neq i} d(i,j)$$

  최소 군집 간 거리 $b(i)$: 현재 군집을 제외한($J \neq I$) 군집 내 데이터끼리의 평균 거리 중 최소값이다. 이는 다른 군집과의 거리가 얼마나 떨어져 있는지를 나타낸다.

  $$b(i) = \min_{J \neq I} \frac{1}{|C_J|} \sum_{j \in C_J} d(i,j)$$

  이제 이 두 값을 이용하여 각 데이터 포인트의 실루엣 지수 $s(i)$를 계산한다.

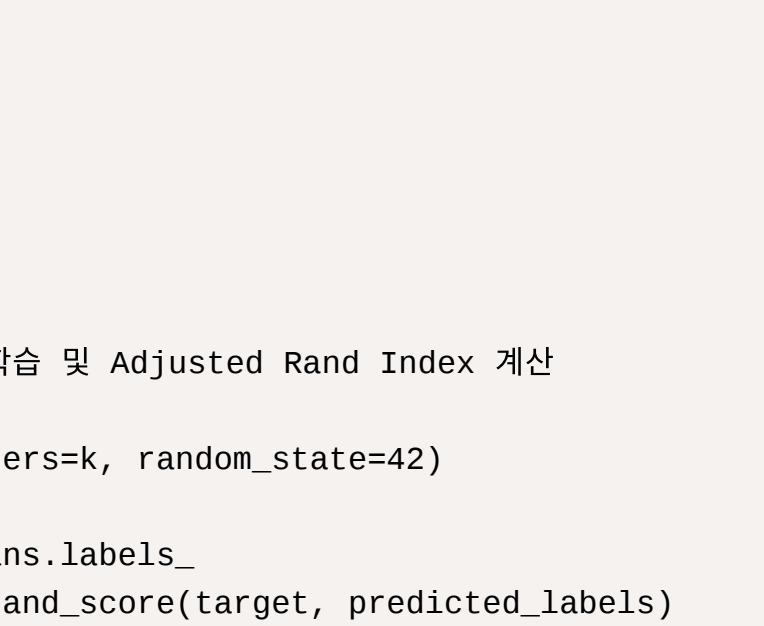
$$s(i) = \begin{cases} \dfrac{b(i) - a(i)}{\max\{a(i), b(i)\}}, & \text{if } |C_I| > 1 \\ 0, & \text{if } |C_I| = 1 \end{cases}$$

- 이 값이 1에 가까우면 군집화가 잘 되었다고 판단되고, -1에 가까우면 잘못된 군집화를 의미한다. 0에 가까운 값은 군집이 서로 겹치거나 중첩되었음을 나타낸다.

코드로 확인

- 붓꽃 데이터를 이용한 k 개수에 따른 rand 지수 확인해보기

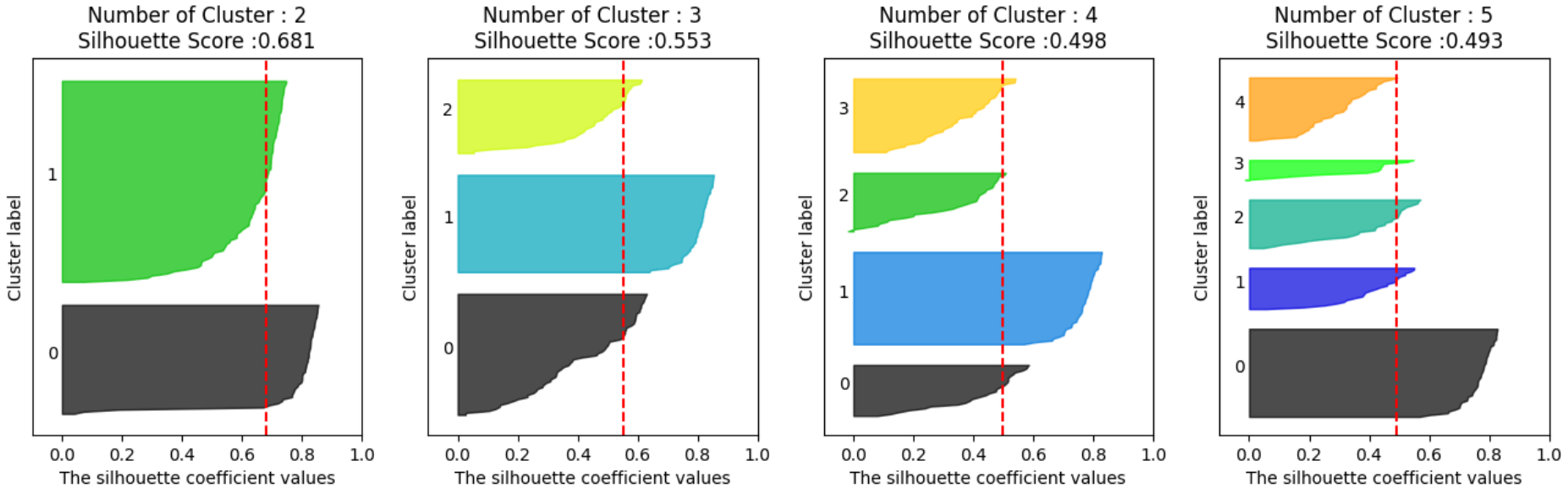
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.metrics import adjusted_rand_score
from sklearn.datasets import load_iris

# iris 데이터셋 로드
iris = load_iris()
data = iris.data
target = iris.target

# K 값 범위 지정
k_values = range(2, 11)

rand_indices = []

# 각 K 값에 대한 KMeans 모델 학습 및 Adjusted Rand Index 계산
for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(data)
    predicted_labels = kmeans.labels_
    rand_index = adjusted_rand_score(target, predicted_labels)
    rand_indices.append(rand_index)

plt.plot(k_values, rand_indices, marker='o')
plt.title('Adjusted Rand Index for Different K Values')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Adjusted Rand Index')
plt.show()
```



rand 지수 상으로는 3 이 가장 좋은 k 값이라고 볼수 있다.

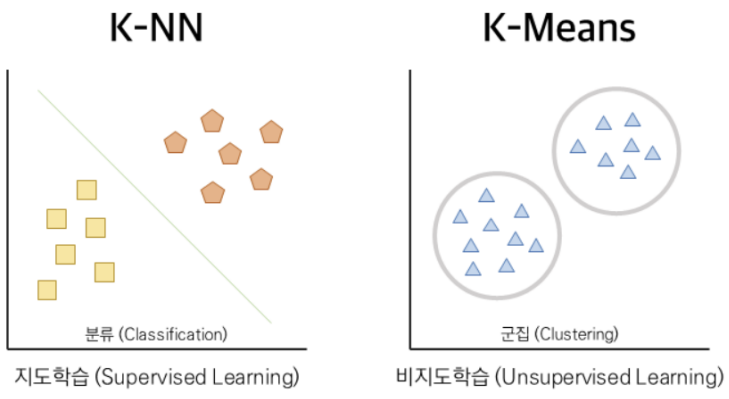
- k 개수에 따른 실루엣 계수 확인하기

```
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="--")
```



실루엣 계수로 보았을 땐, k가 2일 때가 최적의 값이다.

K-최근접 이웃 모델과 K-평균 군집화 모델의 차이를 서술하고 각 모델이 활용될 수 있는 사례를 적어보시오.



✏️ K-최근접 이웃 (K-Nearest Neighbors, KNN)

1. **특징**
   - KNN은 **지도 학습(Supervised Learning)** 알고리즘 중 하나로, 새로운 데이터 포인트를 분류하거나 회귀하는데 사용한다.
   - 주어진 데이터 포인트의 K개의 최근접 이웃을 찾아 다수결 방식으로 분류하거나 평균을 사용하여 회귀한다.
2. **활용 사례**
   - 이상치에 민감하지 않은 경우에 적합하다.
   - 클래스 간 경계가 복잡하고 비선형일 때 효과적이다.
   - 분류 회귀 문제, 단기 교통상황 예측, 추천 시스템, 스팸 메일 필터링등에 사용된다.

✏️ K-평균 군집화 (K-Means Clustering)

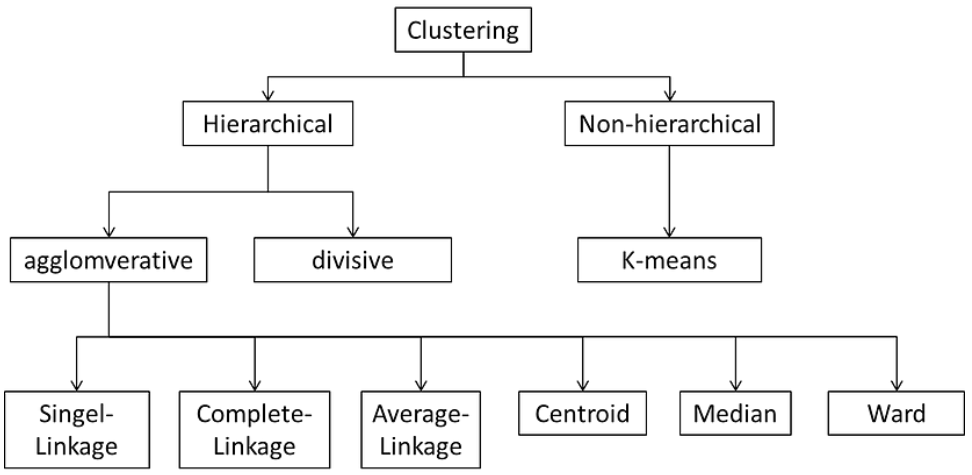
1. **특징**

- K-평균은 **비지도 학습(Unsupervised Learning)** 알고리즘으로, 주어진 데이터를 K개의 군집으로 나눈다.
- 각 군집은 중심이라 불리는 점으로 나타내며, 데이터 포인트는 가장 가까운 중심에 속한다.

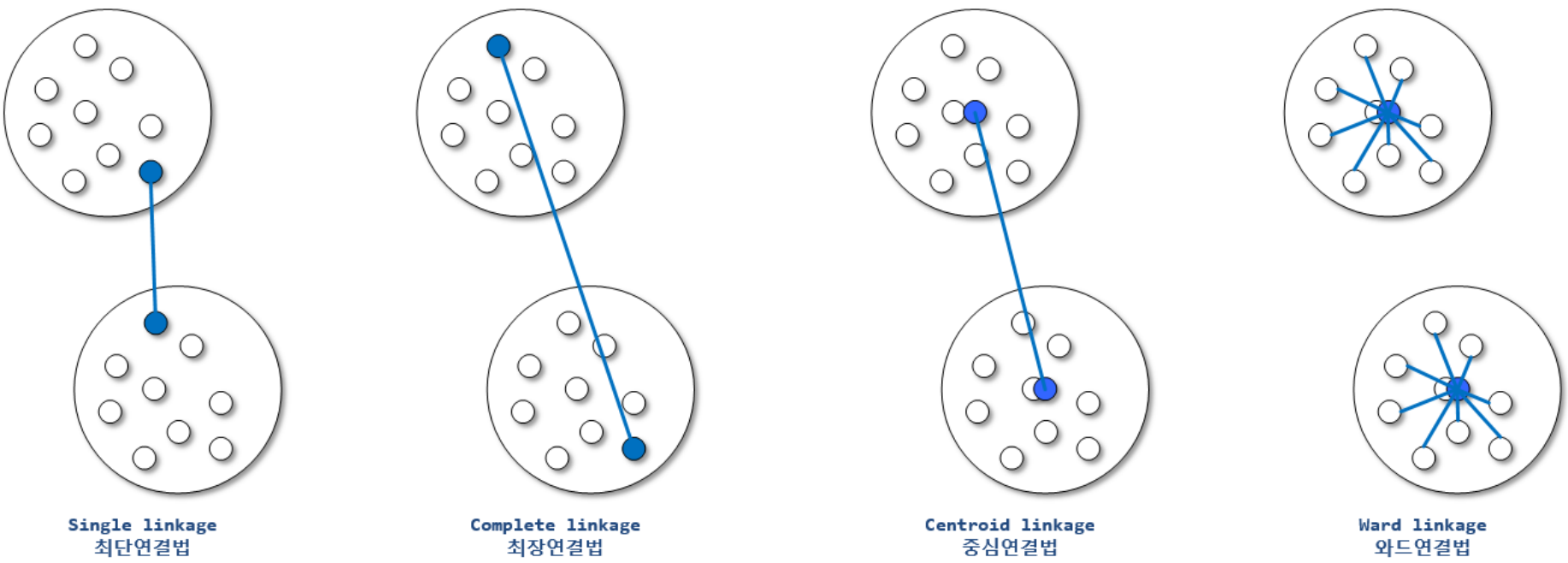
2. **활용 사례**
   - 군집 간의 경계가 명확하고 선형적일 때 효과적이다.
   - 대중교통 이용자 통행패턴 생성, 이상치 탐지, 이미지 분할 등에 사용된다.

계층적 군집화 모델에서 군집 사이의 거리를 정의하는 연결법들을 나열해보고 각 연결법들의 특징에 대해 서술해보시오.

📌 계층적 군집화 vs 비계층적 군집화



특징	계층적 군집	비계층적 군집
구조	계층적 구조, 트리 형태로 군집 형성	각 군집 간 상위/하위 관계 없이 독립적 형성
결과 시각화	덴드로그램을 통해 계층적 결과 시각화	개별 군집 형성, 각 군집 독립적으로 시각화
연결 방식	단일, 완전, 평균 등 다양한 연결 방식 사용	KMeans, DBSCAN 등 다양한 알고리즘 활용
유사도 측정	트리 구조에서 군집 간의 거리 측정	군집 내 데이터 간의 거리나 유사도 측정
적용 분야	계층적 구조 이해, 계층 관계 중요 시 사용	대부분의 군집화 작업에 사용
계산 복잡성	계층 수에 따라 복잡성 증가	일반적으로 낮은 복잡성
군집 개수 결정	군집 수 선택 가능 (덴드로그램 활용)	사용자가 군집 개수 지정



1. 최단 연결법 (Single Linkage)

- 군집 간 원소끼리의 거리를 모두 비교한 후 그중 최소거리를 군집 간 거리로 정의하는 방법
- 비슷한 크기의 여러 군집을 형성하고 싶은 상황에는 사용 지양
- 데이터의 노이즈에 취약하지만 계산 속도 매우 빠름
- 큰 데이터셋에서 계층적 군집화를 수행할 때에는 유용함
- convex set(컨벡스 집합)이 아니라면 상대적으로 성능 우수

2. 최장 연결법 (Complete Linkage)

두 군집 간 원소끼리의 거리를 모두 비교한 후 그중 최대 거리를 군집 간 거리로 정의하는 방법

3. 평균 연결법 (Average Linkage)

두 군집 간 원소끼리의 거리를 모두 비교한 후 평균 거리를 군집 간 거리로 정의하는 방법

4. 중심 연결법 (Centroid Linkage)

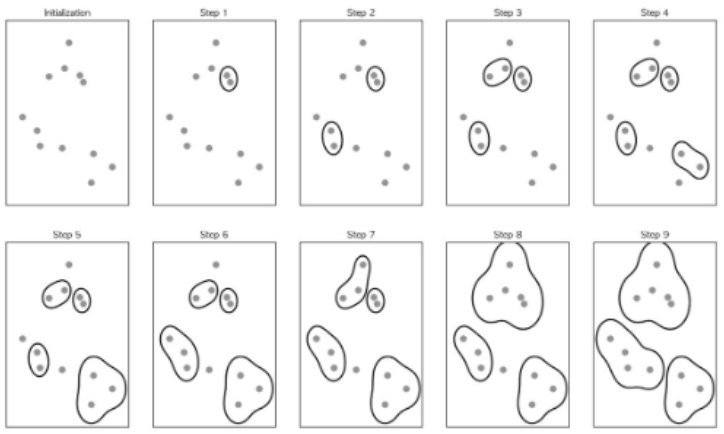
- 각 군집의 중심을 구한 후 중심 사이의 거리를 군집의 거리로 정의하는 방법
- 각 군집의 중심 : 군집 내 원소의 무게중심으로 정의

5. 와드 연결법 (Ward's Linkage)

- 두 군집을 병합했을 때 군집 내 분산의 증가분을 두 군집 사이의 거리로 정의하는 방법
- 군집 내 분산 : 각 군집을 그 군집의 중심으로만 근사했을 때 발생하는 정보의 손실로 해석 가능
- 세부적으로 다양한 변형 정의가 존재
- 비슷한 크기의 여러 군집을 형성하고 싶은 상황에 주로 사용

교재 346페이지에 덴드로그램 시각화 예제를 실습해보면서 각 과정에 대해 설명해보시오.

📌 **병합 군집 (Agglomerative clustering)**



병합군집 알고리즘은 사직할 때 각 포인트를 하나의 클러스터로 지정하고,

그 다음 어떤 종료 조건을 만족할 때까지 가장 비슷한 두 클러스터를 합쳐나간다.

1. **Iris 데이터셋 불러오기**

```
import numpy as np
from matplotlib import pyplot as plt
from scipy.cluster.hierarchy import dendrogram
from sklearn.datasets import load_iris
from sklearn.cluster import AgglomerativeClustering

X = load_iris().data
```

2. **계층적 군집화**

```
model = AgglomerativeClustering(distance_threshold=0, n_clusters=None)
model = model.fit(X)
```

`AgglomerativeClustering`의 인스턴스를 생성.

`distance_threshold=0`은 알고리즘이 모든 쌍의 거리가 0보다 큰 경우까지 클러스터를 형성하도록 한다. `n_clusters=None`은 특정한 클러스터 수가 지정되지 않았다는 것을 나타낸다.

3. **덴드로그램을 위한 개수 계산**

```
counts = np.zeros(model.children_.shape[0])
n_samples = len(model.labels_)
for i, merge in enumerate(model.children_):
    current_count = 0
    for child_idx in merge:
        if child_idx < n_samples:
            current_count += 1  # leaf node
        else:
            current_count += counts[child_idx - n_samples]
    counts[i] = current_count
```

덴드로그램을 생성하기 위해 각 병합에서 각 노드(클러스터)에 속하는 포인트의 수를 계산한다. 이는 계층을 따라 이동하면서 각 병합에서 잎 노드의 수를 세는 것을 포함한다.

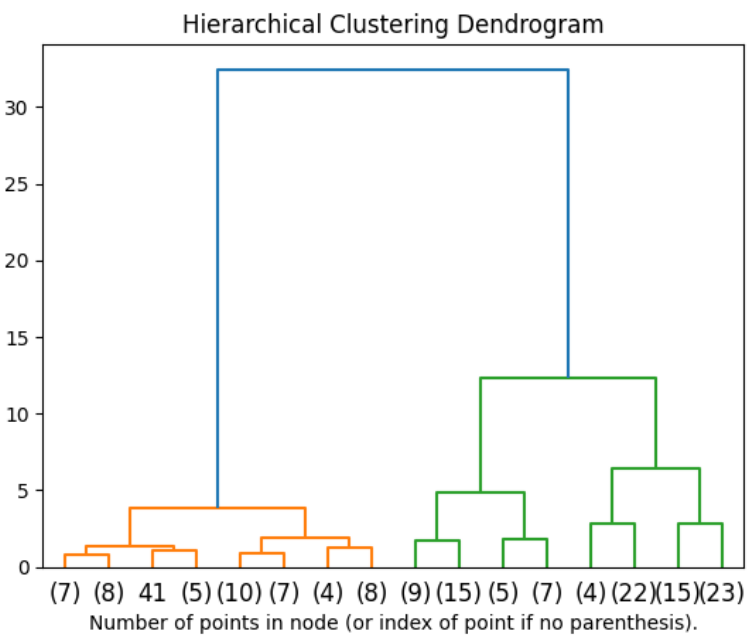
4. **링크지 매트릭스 생성**

```
linkage_matrix = np.column_stack([model.children_, model.distances_, counts]).astype(float)
```

계층적 군집에 관한 정보를 포함하는 링크지 매트릭스를 생성한다. 이는 병합된 클러스터, 그 사이의 거리 및 결과 클러스터에 속하는 포인트 수에 대한 정보를 포함한다.

5. **덴드로그램**

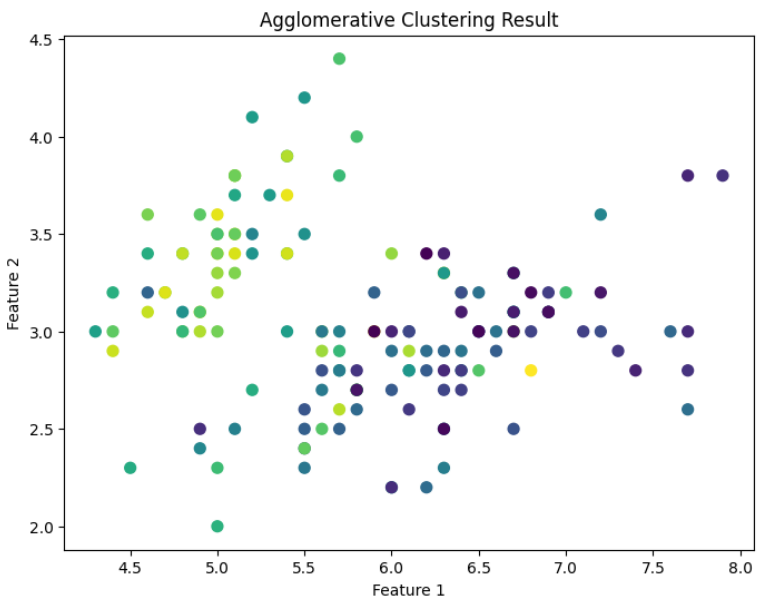
```
dendrogram(linkage_matrix, truncate_mode="level", p=3)
plt.title("Hierarchical Clustering Dendrogram")
plt.xlabel("Number of points in node (or index of point if no parenthesis).")
plt.show()
```



마지막으로 링크지 매트릭스를 사용하여 계층적 군집화 덴드로그램을 시각화. `ncate_mode="level"` 및 `p=3` 매개변수는 덴드로그램의 트리를 특정 레벨에서 잘라내는 데 사용된다.

6. **군집화 결과 시각화**

```
plt.figure(figsize=(8, 6))
plt.scatter(X[:, 0], X[:, 1], c=model.labels_, cmap='viridis', s=50)
plt.title('Agglomerative Clustering Result')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```



군집화 개수를 지정하지 않은 상태에서 알고리즘이 자동으로 군집화한 결과이다.

7. **군집화 개수 확인**

```
print("Number of clusters:", model.n_clusters_)
```

결과

```
Number of clusters: 150
```

150 개의 군집화가 이루어진 것을 확인할 수 있다.

이전 5장에서 릿지 회귀의 유일해는 $\hat{\omega}_{\text{Ridge}} = (X^T X + \alpha I)^{-1} X^T \mathbf{y}$ *임을 알 수 있었다. 이를 X의 특이값 분해를 이용해 다시 나타내면,* $X\hat{\omega}_{\text{Ridge}} = U\sum(\sum^2 + \alpha I)^{-1}\sum U^T \mathbf{y}$이고, 규제가 없는 최소제곱법 모델의 해는 $X\hat{\omega} = UU^T\mathbf{y}$이다. 이를 통해 PCA 모델과 릿지회귀에는 어떠한 관련성이 있는지 설명해보이시오.

릿지 회귀의 해에서 α는 규제 파라미터이다. α를 통해 특이값(주성분)에 대한 가중치를 조절해 모델 과적합을 방지 및 모델의 일반화 성능을 향상시킨다.

규제가 없는 최소제곱법 모델의 해에서 U는 입력 데이터X의 특이 벡터이고 이는 원래 데이터 공간에 대한 선형 변환이다.

PCA는 데이터 분산을 최대화하는 주성분을 찾는 방법으로 원래 데이터 공간에서 새로운 주성분 공간으로의 매핑(차원 축소)이 이뤄진다.

PCA의 이 변환과 규제없는 OLS 모델의 선형 변환이 굉장히 유사하다.
따라서 릿지회귀의 해는 PCA의 주성분에 대한 가중치를 조절하는 형태로 나타날 수 있다는 점에서 관련성이 있다고 할 수 있다.

차원 축소기법인 MDS, Isomap, LLE, t-SNE의 특징에 대해 서술해보시오.

1. MDS(multidimensional scaling, 다차원 척도법)

- 주어친 고차원에서의 샘플 간 거리가 저차원에서도 최대한 보존되도록 변환하는 선형 차원 축소 기법
- 샘플 간의 거리를 모두 계산한 후 이 정보를 활용
- 거리 행렬을 최대한 보족하는 저차원의 좌표계 계산

- sklearn.manifold,MDS 클래스로 구현

2. Isomap

- MDS와 비슷 BUT 샘플 사이의 거리 행렬을 구하는 방법만을 달리한 비선형 차원 축소 기법
- 거리 행렬 구할 때, 유클리드 거리 대신 데이터에 내재적인 매니폴드를 가정하고 그 매니폴드상에서의 거리를 최대한 보존하도록 변환 수행
- 샘플별로 주위 샘플과의 거리를 나타내는 그래프 구축 -> 두 점 사이의 거리는 구축한 그래프에서 최단 거리를 계산하여 얻음
- 위 과정을 모든 샘플에 반복하여 거리 행렬 얻은 후 MDS와 마찬가지의 방식으로 차원 축소
- sklearn.manifold.Isomap 클래스로 구현

3. LLE(locally linear embeding)

- 비선형 차원 축소 기법
- Isomap과 마찬가지로, 각 샘플의 이웃에 대한 정보 활용해 차원 축소 진행
- 각 샘플의 최근접 이웃을 찾은 후 이웃 샘플로 해당 샘플을 근사하거나 재구성
- 마지막으로 재구성한 정보 이용해 저차원으로 축소 진행
- sklearn.manifold.LocallyLinerarEmbeding 클래스로 구현

4. t-SNE(t-distributed stochastic neighbor embedding)

- 확률적으로 이웃을 선택한다고 가정 cf) LLE에서는 샘플별로 고정된 이웃에 대한 정보 사용
- 가까운 샘플일수록 이웃으로 선택될 호가률이 높지만 거리가 먼 샘플이더라도 이웃으로 선택될 확률이 있다고 봄
- 원 데이터 공간에서 한 샘플이 다른 샘플을 이웃으로 선택할 확률 분포가 저차원 공간에서도 유지되도록 차원 축소 진행
- 쿨백-라이블러 발산 개념 사용
- sklearn.manifold.TSNE 클래스로 구현