Unsupervised Learning - Part 2

Machine Learning for Engineering Applications

Fall 2023

Hierarchical Tree

- Hierarchical clustering algorithms is that it allows us to plot → help with the interpretation of the results by creating meaningful taxonomies.

- Another advantage of hierarchical → do not need to specify the number of clusters up front.

- Two types of hierarchical:
  - Agglomerative
  - Divisive

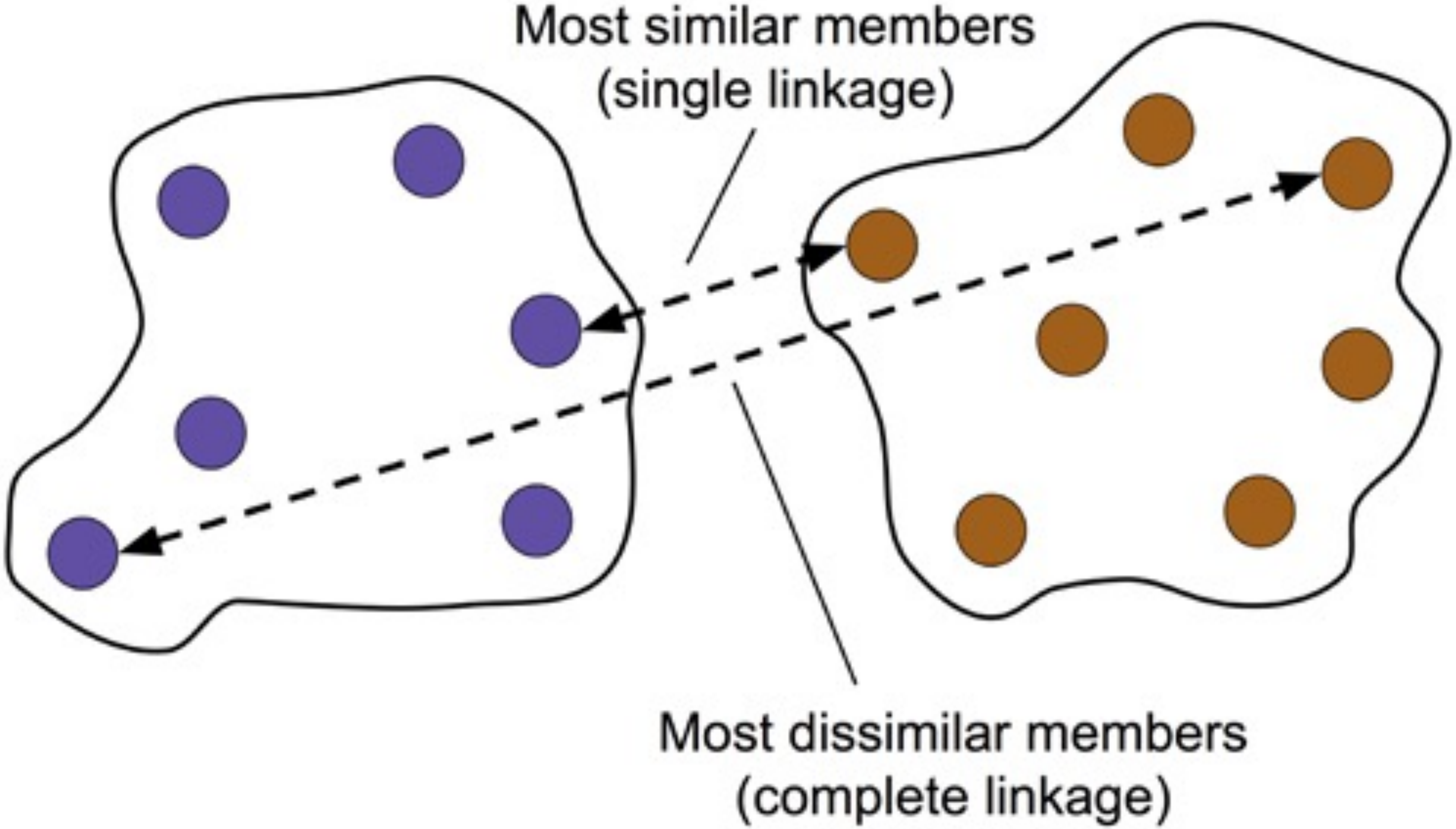
- Divisive
  - Start with one cluster that wraps all of the samples
  - Iteratively split the cluster into smaller clusters until each cluster only contains one sample
- Agglomerative (*Main focus*)
  - Start with each sample as an individual cluster
  - Merge the closest pairs of clusters until only one cluster remains

- Agglomerative Types
  - Single Linkage
  - Complete Linkage

- Single Linkage:
  - Compute the distances between the most similar members for each pair of clusters
  - Merge the two clusters for which the distance between the most similar members is the smallest

- Agglomerative Types
  - Single Linkage
  - Complete Linkage

- Complete Linkage (*main focus*):
  - Opposite of the single linkage
  - Compare the most dissimilar members to perform the merge


Most similar members
(single linkage)
Most dissimilar members
(complete linkage)

- Agglomerative – Complete Linkage Steps

1. Compute the distance matrix of all samples
2. Represent each data point as a singleton cluster
3. Merge the two closest clusters based on the distance between the most dissimilar (distant) members
4. Update the similarity matrix
5. Repeat steps 2-4 until one single cluster remains

- Example code

```
import pandas as pd
import numpy as np

np.random.seed(123)
variables = ['X', 'Y', 'Z']
labels = ['ID_0','ID_1','ID_2','ID_3','ID_4']
X = np.random.random_sample([5,3])*10

df = pd.DataFrame(X, columns=variables, index=labels)
df
```

- Example code

```
df = pd.DataFrame(X, columns=variables, index=labels)
```

	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

- Step 1: Distance Matrix

```
from scipy.spatial.distance import pdist, squareform

row_dist = pd.DataFrame(squareform(
          ... pdist(df, metric='euclidean')),
          ... columns=labels, index=labels)

row_dist
```

- Step 1: Distance Matrix

	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

- Step 2-5

```
from scipy.cluster.hierarchy import linkage

row_clusters = linkage(df.values,
                  ... method='complete’,
                  ... metric='euclidean’)
pd.DataFrame(row_clusters,
             ... columns=['row label 1’,
             ... 'row label 2’,
             ... 'distance’,
             ... 'no. of items in clust.’],
             ... index=['cluster %d' %(i+1) for i in
             ... range(row_clusters.shape[0])])
```

Step 2-5

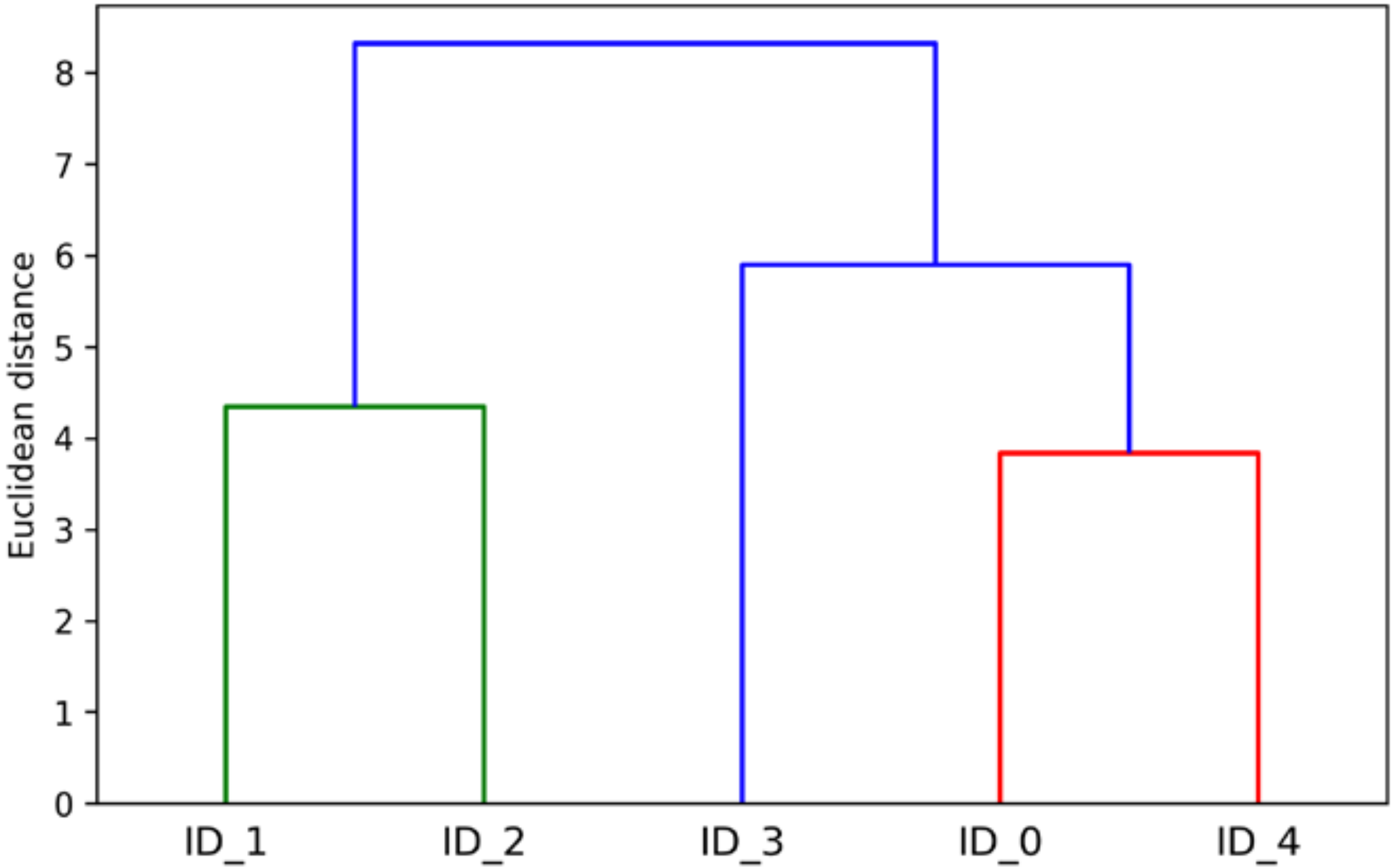
- The 1st & 2nd columns denote the most dissimilar members in each cluster
- The 3rd column reports the distance between those members.
- The 4th column returns the count of the members in each cluster

	row label 1	row label 2	distance	no. of items in clust.
cluster 1	0.0	4.0	3.835396	2.0
cluster 2	1.0	2.0	4.347073	2.0
cluster 3	3.0	5.0	5.899885	3.0
cluster 4	6.0	7.0	8.316594	5.0

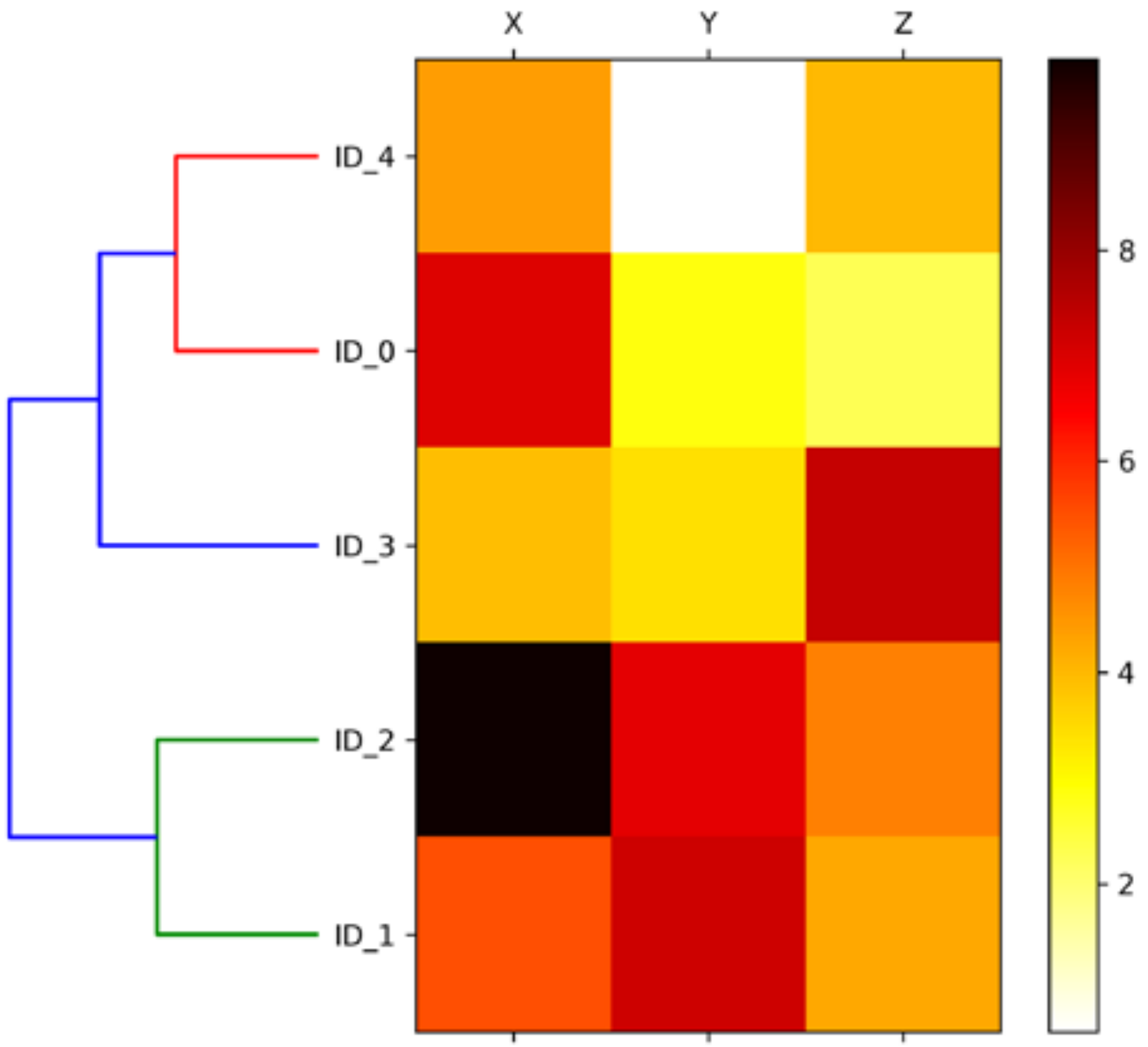
- Dendrogram figure

```
from scipy.cluster.hierarchy import dendrogram
from scipy.cluster.hierarchy import set_link_color_palette
set_link_color_palette(['black'])

row_dendr = dendrogram(row_clusters,
... labels=labels,
... color_threshold=np.inf )

plt.tight_layout()
plt.ylabel('Euclidean distance')
plt.show()
```

- Dendrogram figure



- Dendrogram Heatmap (*code in the textbook*)



SKLearn Agglomerative

- Agglomerative Complete Linkage (3 groups)

```
from sklearn.cluster import AgglomerativeClustering

ac = AgglomerativeClustering(n_clusters=3, # 0 1 2
                         ... affinity='euclidean’,
                         ... linkage='complete’)

labels = ac.fit_predict(X)

print('Cluster labels: %s' % labels)


Cluster labels: [1 0 0 2 1]
```

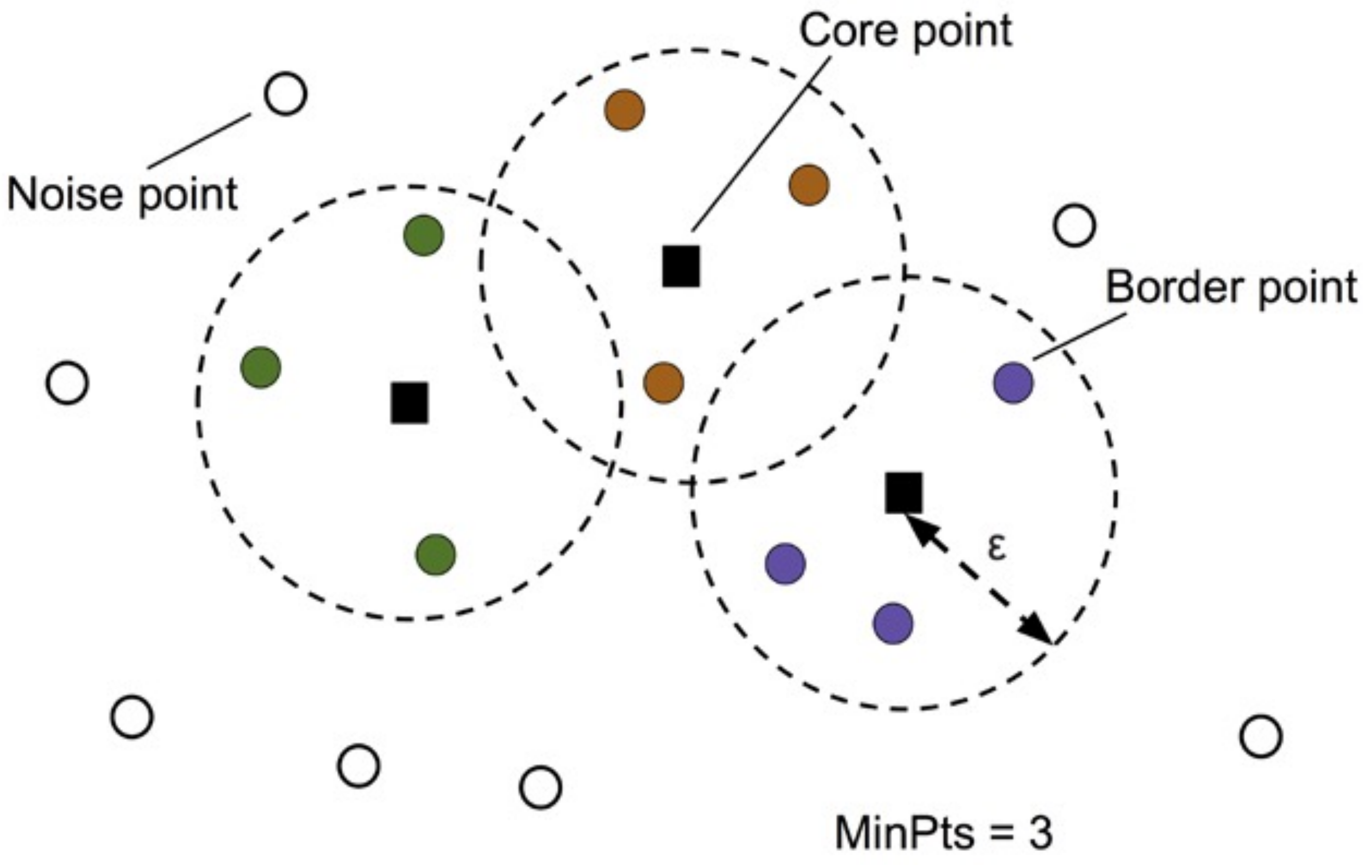
- Agglomerative Complete Linkage (2 groups - Prunning)

```
from sklearn.cluster import AgglomerativeClustering

ac = AgglomerativeClustering(n_clusters=2, # 0 1
                            ... affinity='euclidean',
                            ... linkage='complete')

labels = ac.fit_predict(X)

print('Cluster labels: %s' % labels)


Cluster labels: [0 1 1 0 0]
```

Density-based Spatial Clustering of Apps with Noise (DBSCAN)

- Density-based clustering assigns cluster labels based on dense regions of points

- The notion of density is defined as the number of points within a specified radius ε



- Labels are assigned

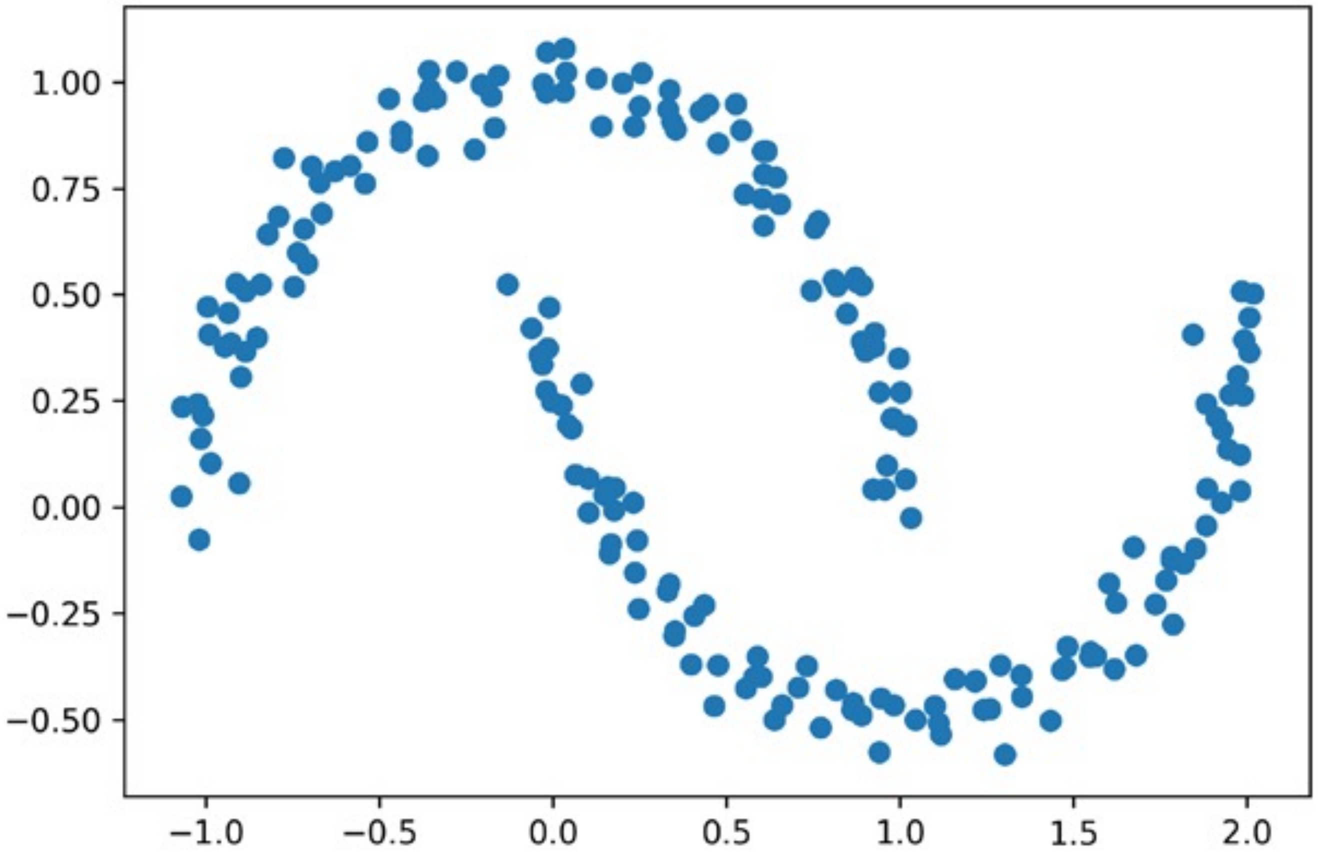
- **A point** is considered a core point if at least a specified number (MinPts) of neighboring points fall within the specified radius ε
- **A border point** is a point that has fewer neighbors than *MinPts* within ε, but lies within the ε radius of a core point
- **All other points** that are neither core nor border points are considered noise points

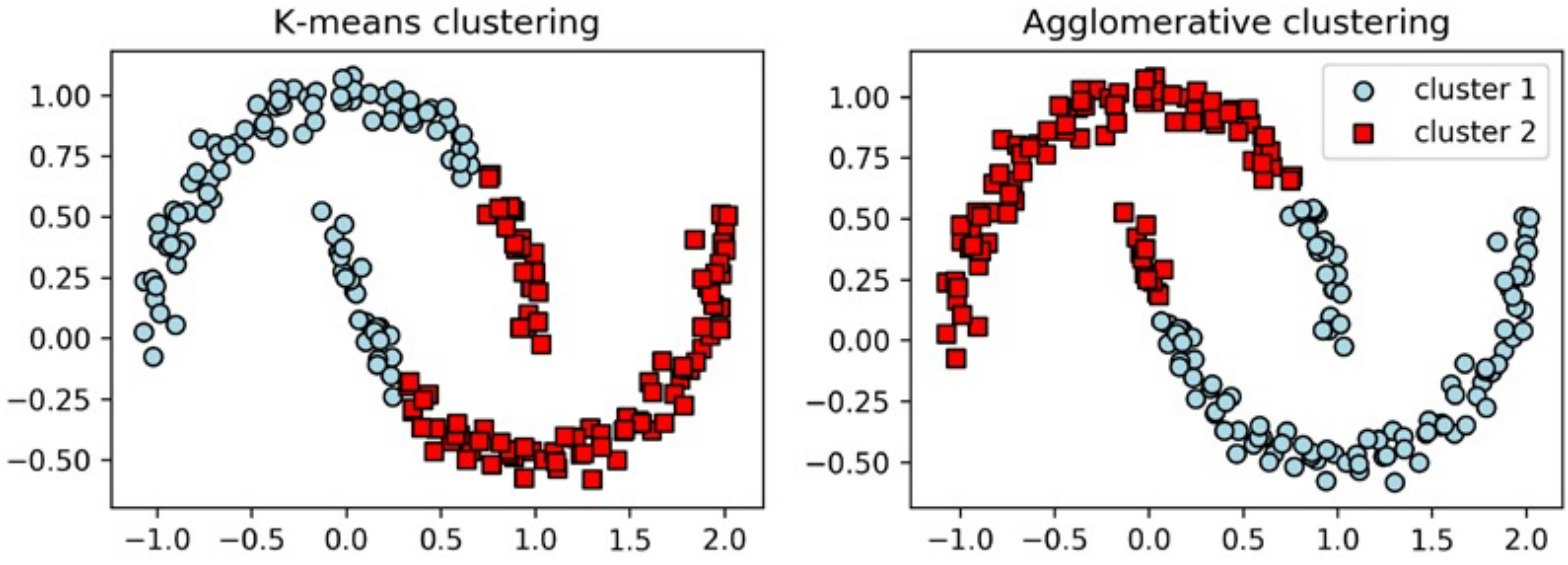
- After labeling

- Form a separate cluster for each core point or connected group of core points
  - Core points are connected if they are no farther away than ε

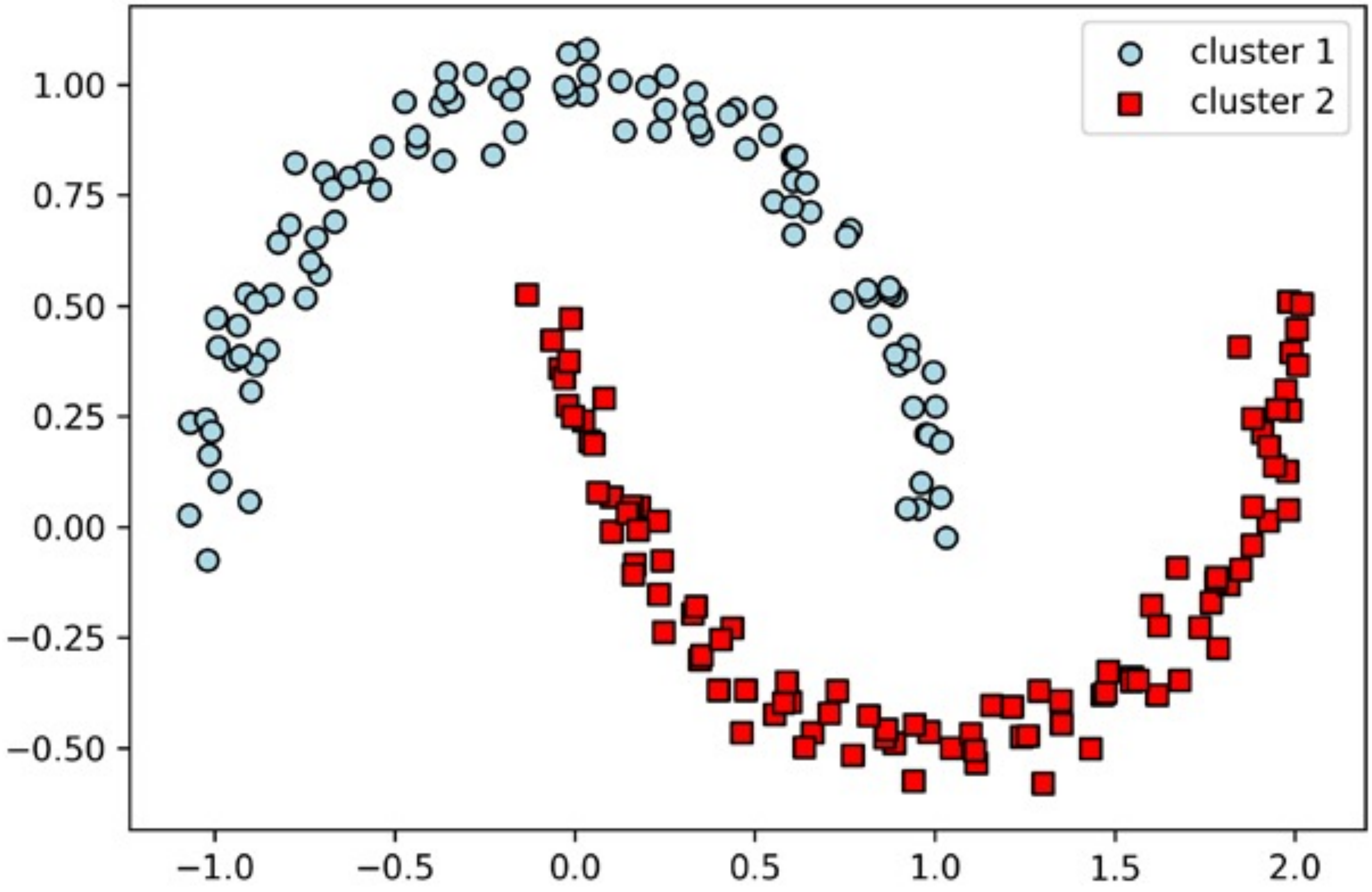
- Assign each border point to the cluster of its corresponding core point.

```
from sklearn.datasets import
make_moons

X, y =
make_moons(n_samples=200,
      ... noise=0.05,
      ... random_state=0)

plt.scatter(X[:,0], X[:,1])
plt.show()
```




K-means clustering
Agglomerative clustering
cluster 1
cluster 2
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

```
from sklearn.cluster import DBSCAN

db = DBSCAN(eps=0.2, min_samples=5, metric='euclidean’)

y_db = db.fit_predict(X)

plt.scatter(X[y_db==0,0], X[y_db==0,1], c='lightblue’,
          ... edgecolor='black’, marker='o', s=40,
          ... label='cluster 1’)

plt.scatter(X[y_db==1,0], X[y_db==1,1], c='red’,
          ... edgecolor='black', marker='s', s=40,
          ... label='cluster 2')
plt.legend()
plt.show()
```


cluster 1
cluster 2
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

- Summary of unsupervised methods
  - Create to quickly see how they get classified without any labels
  - Problems can occur is there are a lot of features for all 3 techniques
  - Parameter tunning will have to be evaluated in all techniques
  - Knowing the number of cluster (classes) will be a huge advantage, but not a requirement.