PATTERN RECOGNITION USING PYTHON

Clustering

Wen-Yen Hsu
Dept Electrical Engineering
Chang Gung University, Taiwan

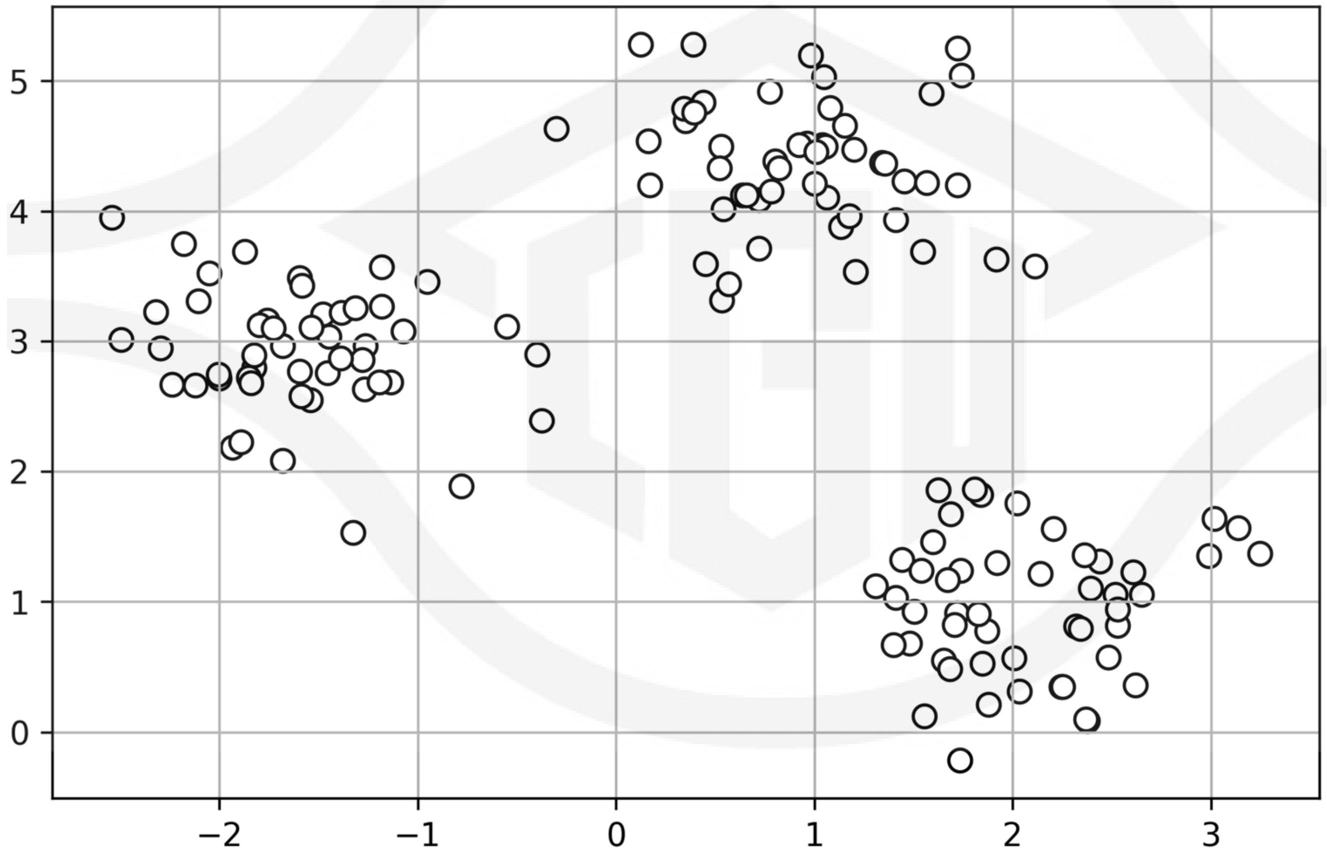
2019-Spring

Organize Data into Meaningful Structures

- Finding centers of similarity using the popular k-means algorithm
- Taking a bottom-up approach to building hierarchical clustering trees
- Identifying arbitrary shapes of objects using a density-based clustering approach

Working with Unlabeled Data

- Unsupervised learning



Four Steps of k-means Algorithm

- Randomly pick $\boldsymbol{k}$ centroids from the sample points as initial cluster centers
- Assign each sample to the nearest centroid $\boldsymbol{\mu}^{(j)}$, $j=1,\ldots,k$
- Move the centroids to the center of the samples that were assigned to it
- Repeat steps 2 and 3 until the cluster assignments do not change or a user-defined tolerance or maximum number of iterations is reached

Measure Similarity between Objects

- **Squared Euclidean Distance** between two points $\mathbf{x}$ and $\mathbf{y}$ in m-dimensional space

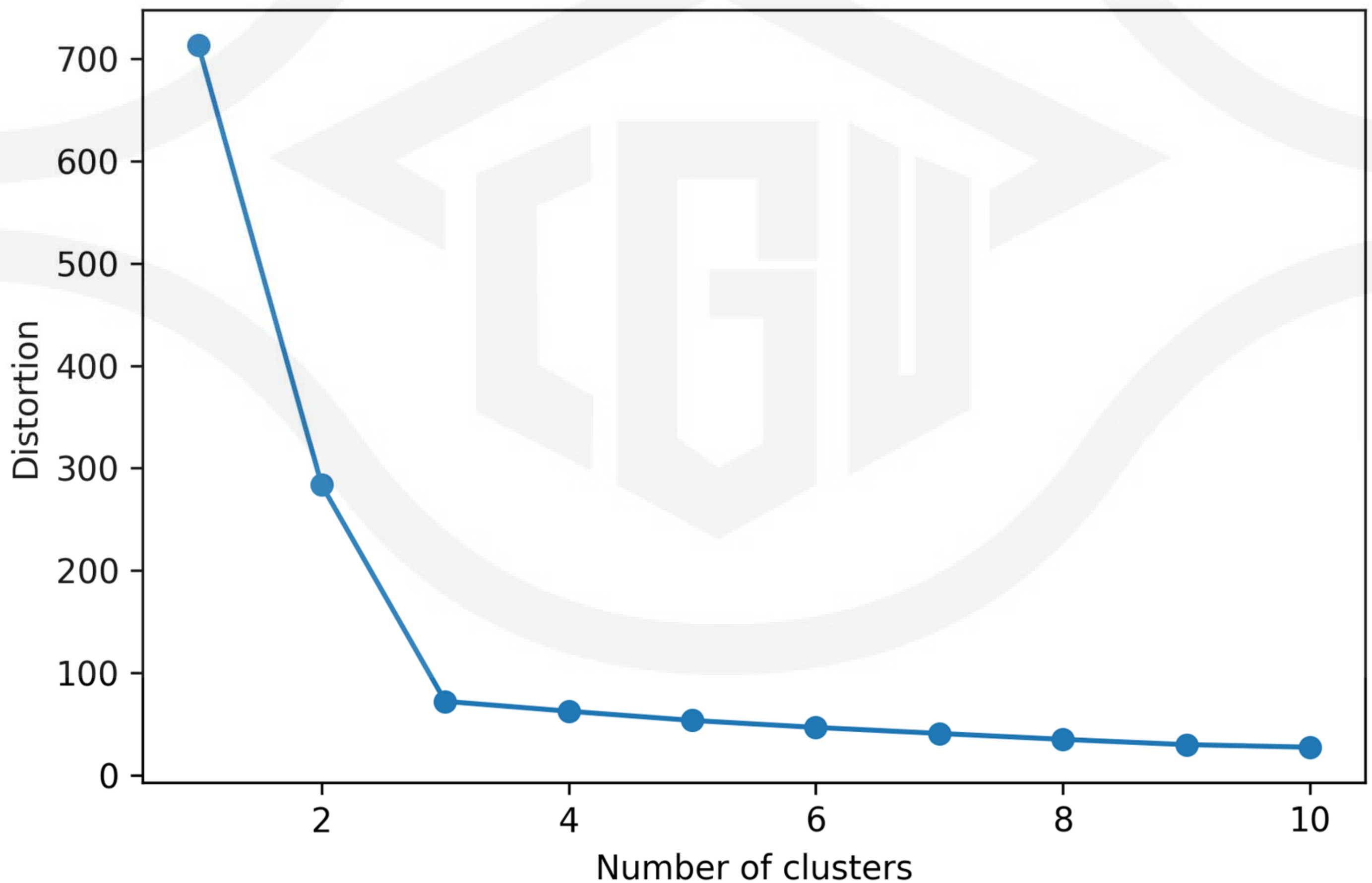
$$d(\boldsymbol{x}, \boldsymbol{y})^2 = \sum_{j=1}^{m} \left(x_j - y_j \right)^2 = \left\| \boldsymbol{x} - \boldsymbol{y} \right\|_2^2$$

- **Cluster Inertia:** an iterative approach for minimizing the within-cluster **Sum of Squared Errors (SSE)**

$$SSE = \sum_{i=1}^{n} \sum_{j=1}^{k} w^{(i,j)} \left\| \boldsymbol{x}^{(i)} - \boldsymbol{\mu}^{(j)} \right\|_2^2$$

Find the Optimal Number *k* of Clusters

- Compare the performance of different k-means clusters based on the within-cluster SSE



k-mean Clustering

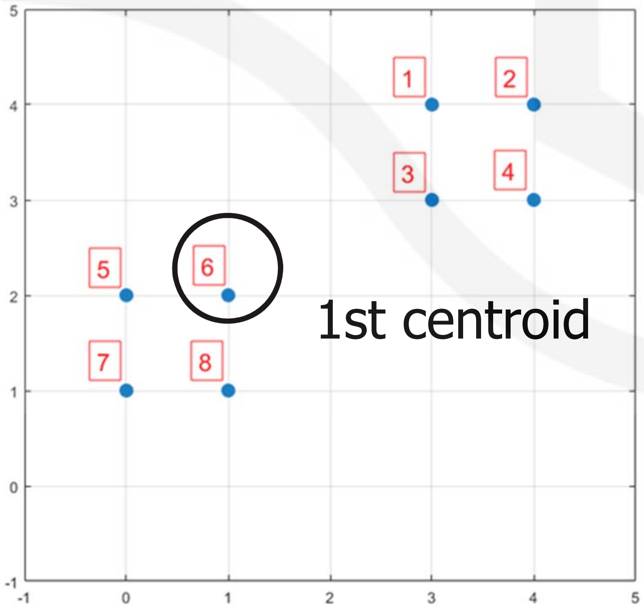
- Finding centers of similarity

```
from sklearn.cluster import KMeans
km = KMeans(n_clusters=3,
            init='random',
            n_init=10,
            max_iter=300,
            tol=1e-04,
            random_state=0)
y_km = km.fit_predict(X)
print(y_km)
print('Distortion: %.2f' % km.inertia_)
```

k-means++

- Similar to the k-mean algorithm but strategically improving the select method of the next centroid

- To randomly select the next centroid $\boldsymbol{\mu}^{(\mathrm{p})}$, use a weighted probability distribution equal to $\dfrac{d\left(\boldsymbol{\mu}^{(\mathrm{p})}, \mathbf{M}\right)^2}{\sum_i d\left(\boldsymbol{x}^{(i)}, \mathbf{M}\right)^2}$



Performing k-means++

- Randomly generate a random number between 0 and 1, to determine which interval it belongs to, then the number corresponding to the interval is the second centroid selected.

	1	2	3	4	5	6	7	8
d	$2\sqrt{2}$	$\sqrt{13}$	$\sqrt{5}$	$\sqrt{10}$	1	0	$\sqrt{2}$	1
d^2	8	13	5	10	1	0	3	1
P	0.2	0.325	0.125	0.25	0.025	0	0.05	0.025
ΣP	0.2	0.525	0.65	0.9	0.925	0.925	0.975	1

2nd centroid

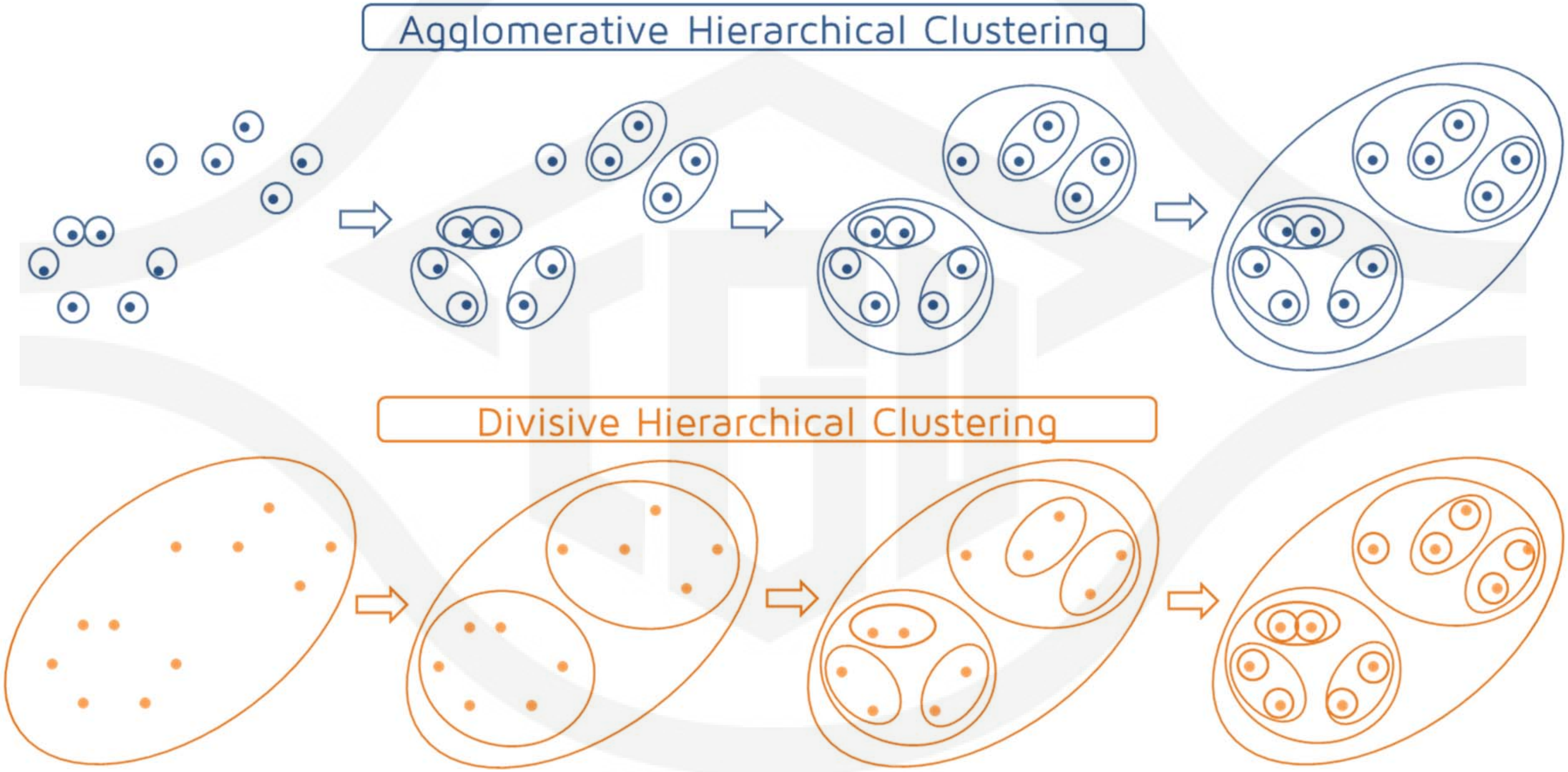
1st centroid

K-mean++ Clustering

- The next centroid is far from the previous one

```
from sklearn.cluster import KMeans
km = KMeans(n_clusters=3,
            init='k-mean++',
            n_init=10,
            max_iter=300,
            tol=1e-04,
            random_state=0)
y_km = km.fit_predict(X)
print(y_km)
print('Distortion: %.2f' % km.inertia_)
```

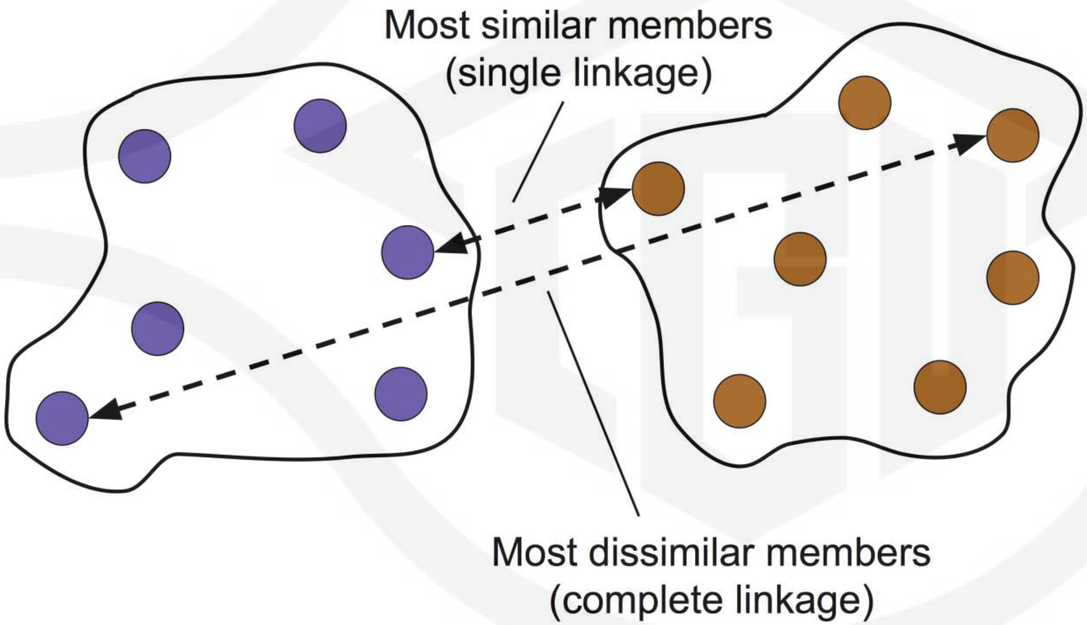
Hierarchical Clustering



https://quantdare.com/hierarchical-clustering/agglodivhierarclustering/

Agglomerative Hierarchical Clustering

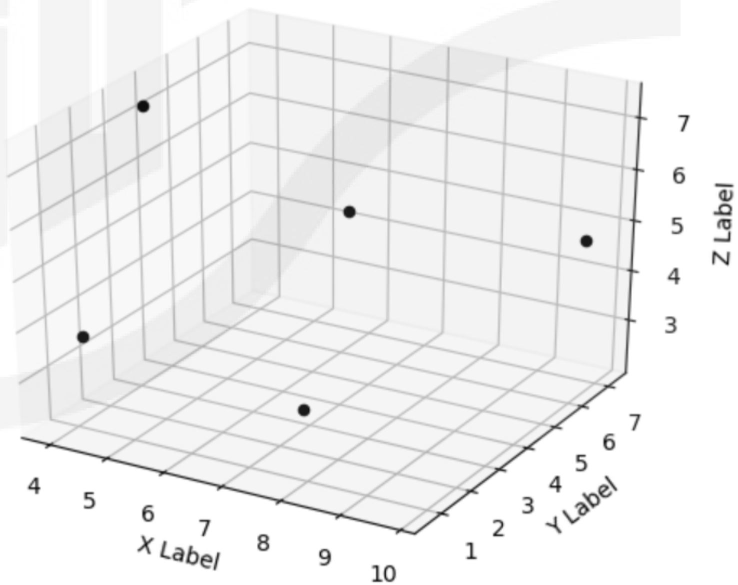
- Merge the two clusters for which the distance between the most similar members is the smallest



- Merge the two closest clusters based on the distance between the most dissimilar (distant) members

Randomly Generate 5 Unlabeled Data

	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



Compute the Distance Matrix and Clustering

- Distance matrix of all samples

	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

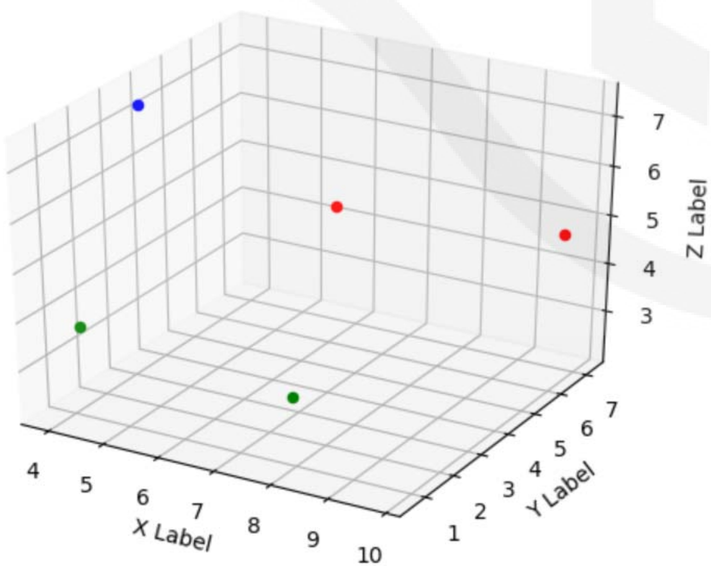
- Clustering approach

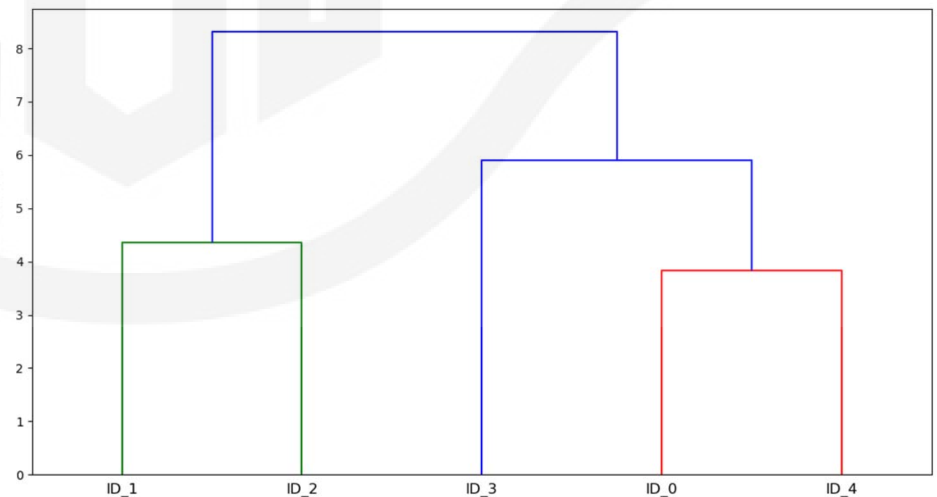
	row label 1	row label 2	distance	no. of items in clust.
cluster 1	0.0	4.0	3.835396	2.0
cluster 2	1.0	2.0	4.347073	2.0
cluster 3	3.0	5.0	5.899885	3.0
cluster 4	6.0	7.0	8.316594	5.0

Agglomerative Clustering via sklearn

- Bottom-up approach

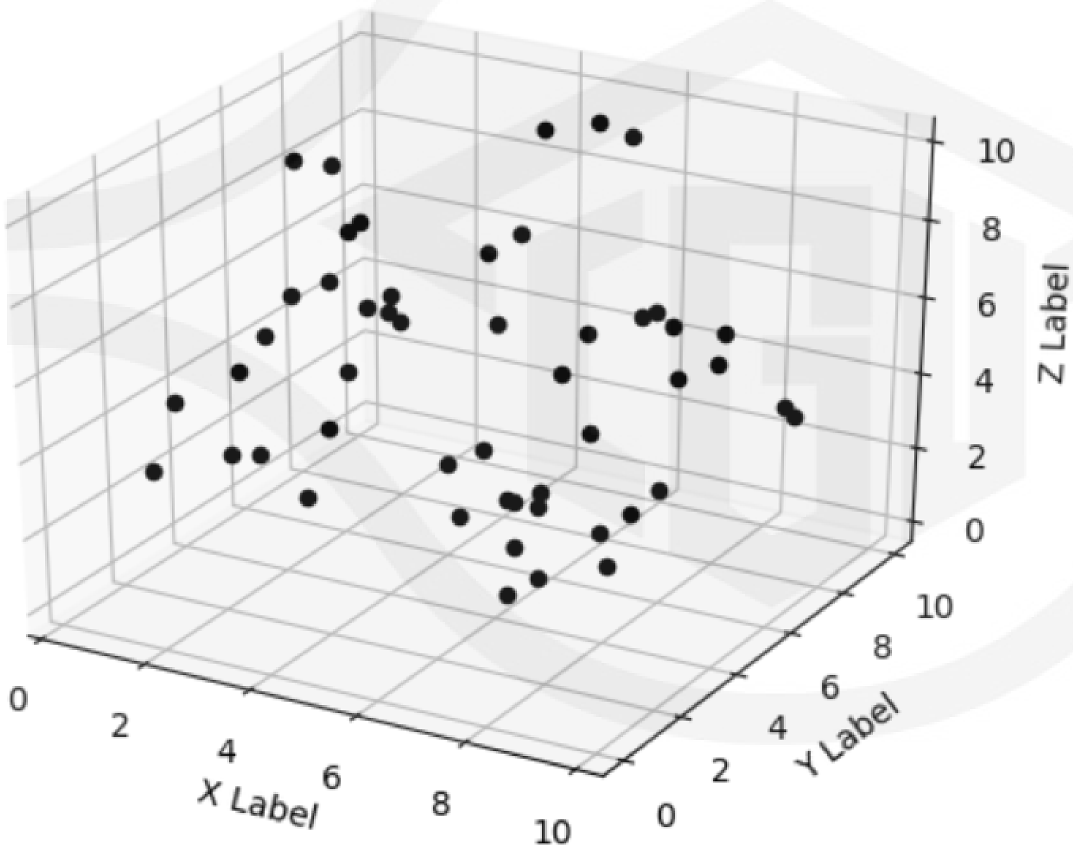
```
from sklearn.cluster import AgglomerativeClustering
ac = AgglomerativeClustering(n_clusters=3,
                              affinity='euclidean',
                              linkage='complete')
cluster_labels = ac.fit_predict(X)
print('Cluster labels: %s' % cluster_labels)
```





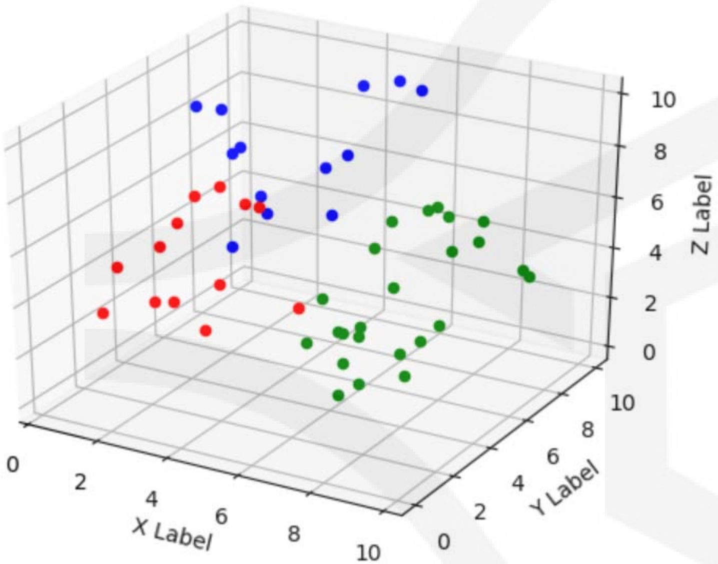
Difference Agglomerative Linkage

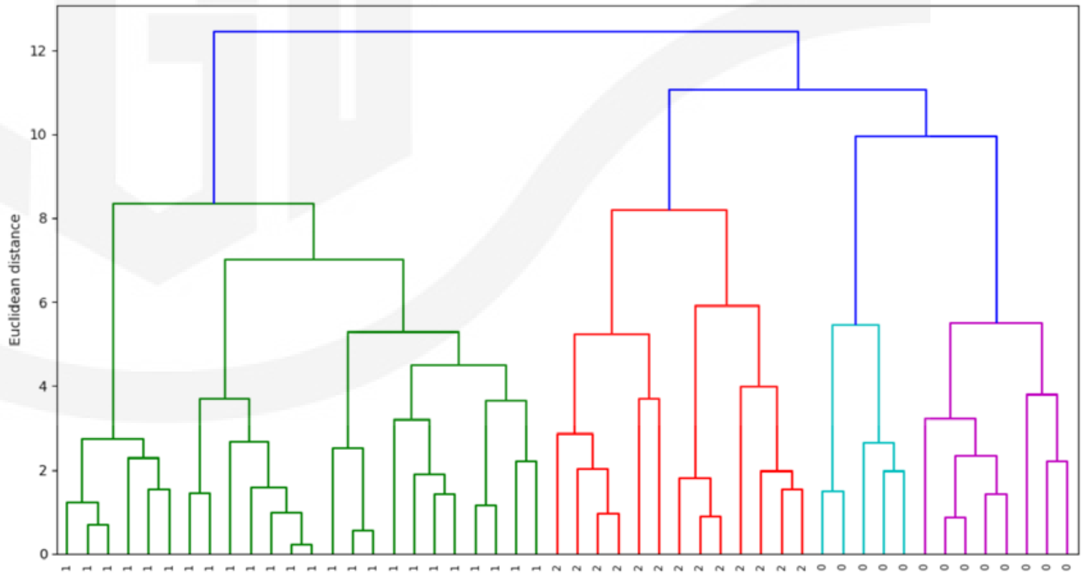
- Randomly Generate 50 Unlabeled Data



Complete Linkage

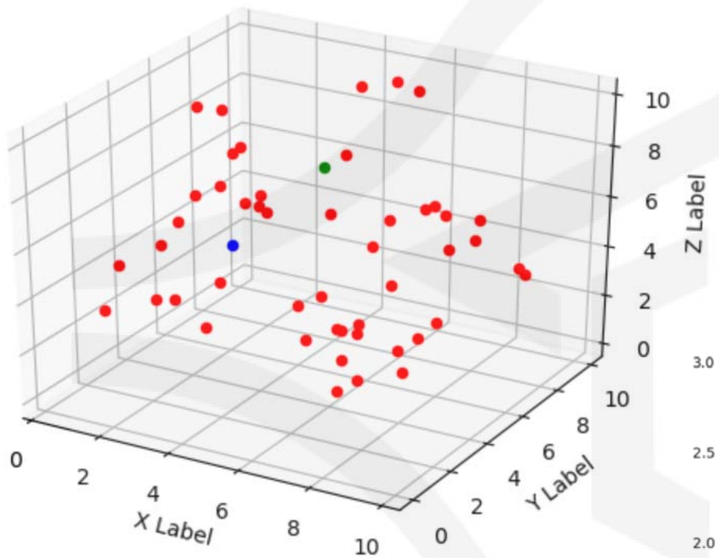
- Balance clustering

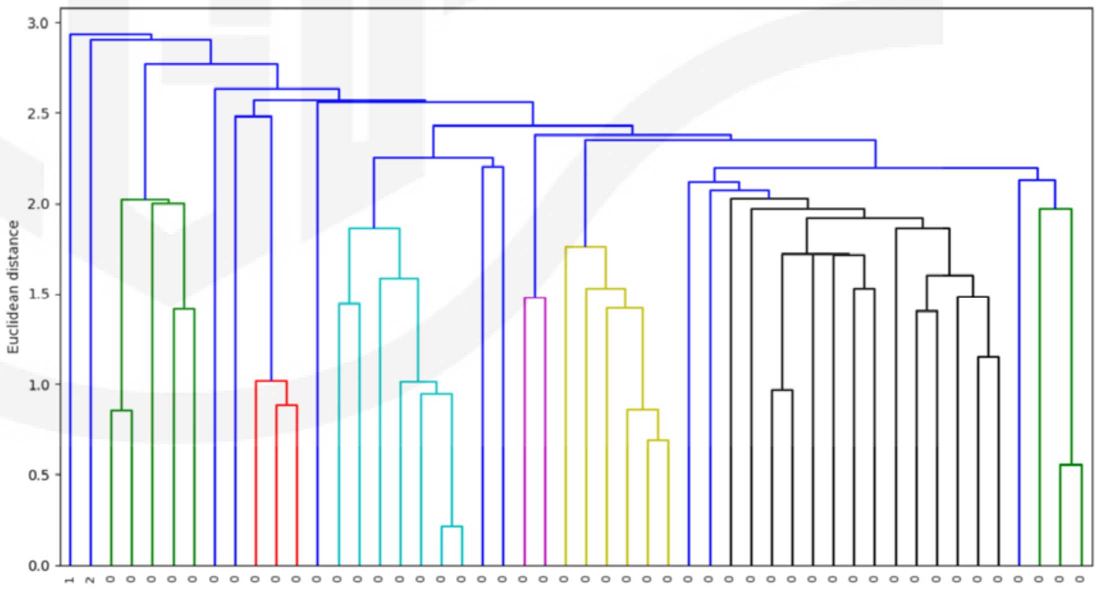




Single Linkage

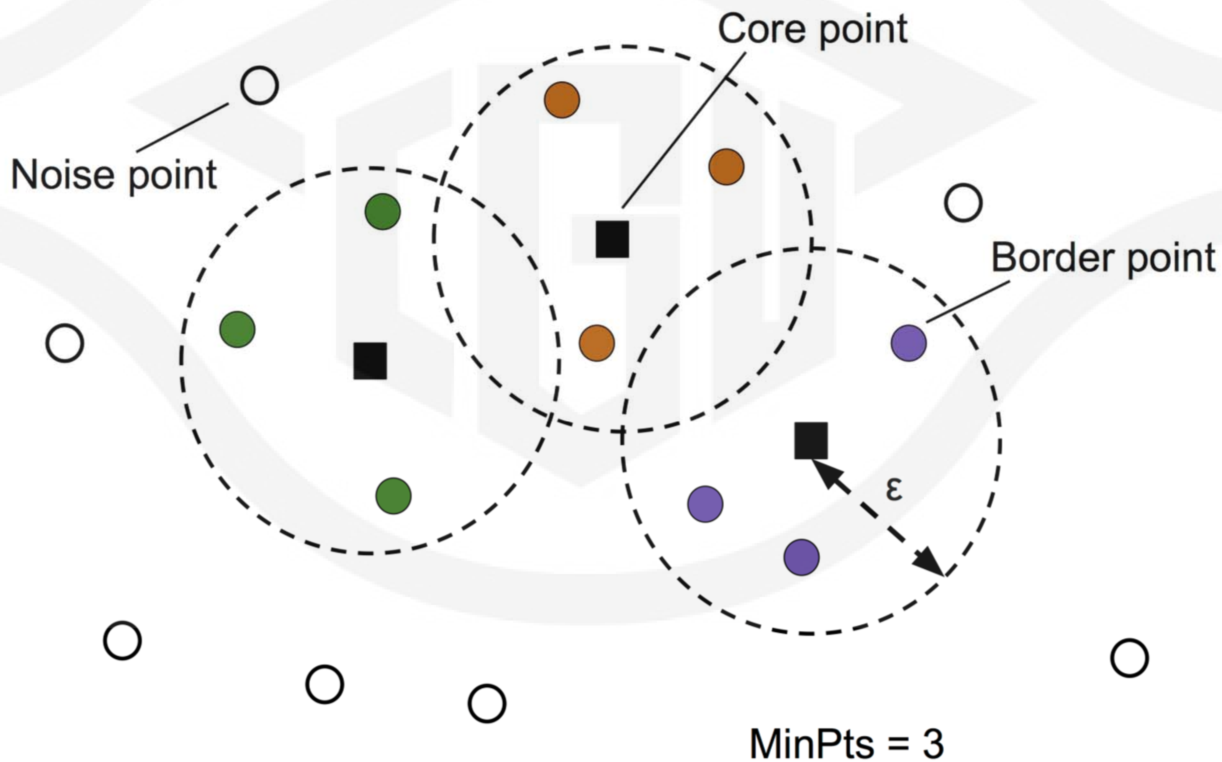
- Unbalance clustering





DBSCAN

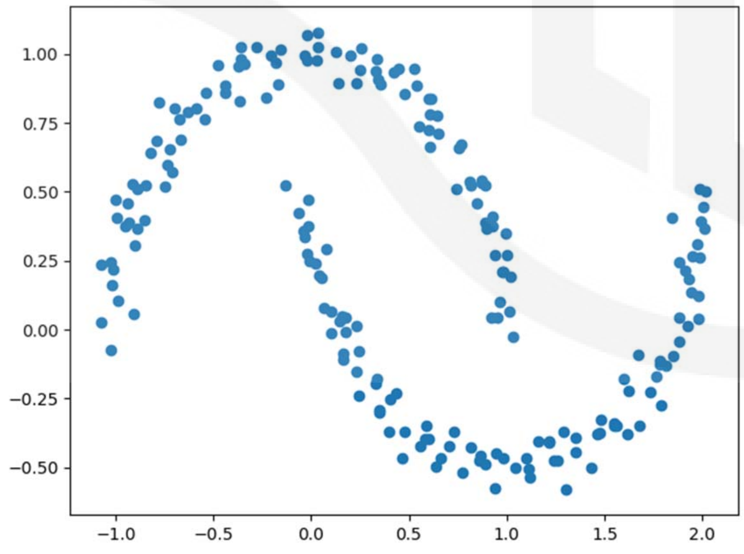
- Density-based Spatial Clustering of Applications with Noise
- No need to select k but need to define ε and MinPts

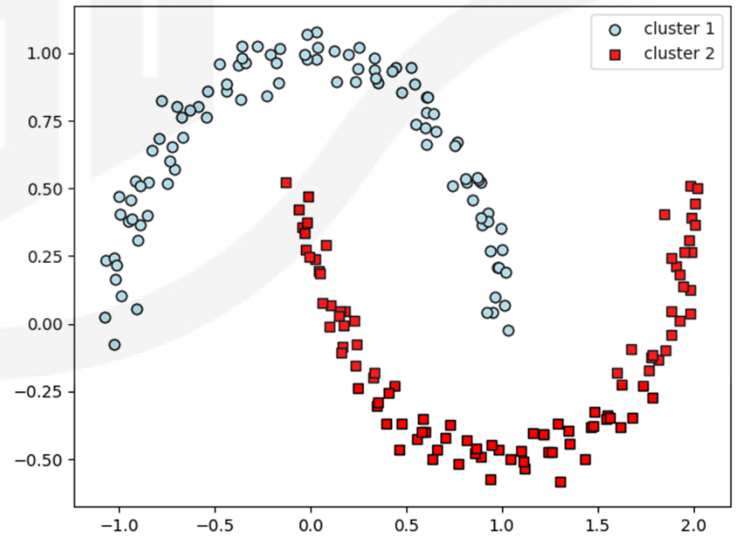


DBSCAN via sklearn

- Density-based clustering

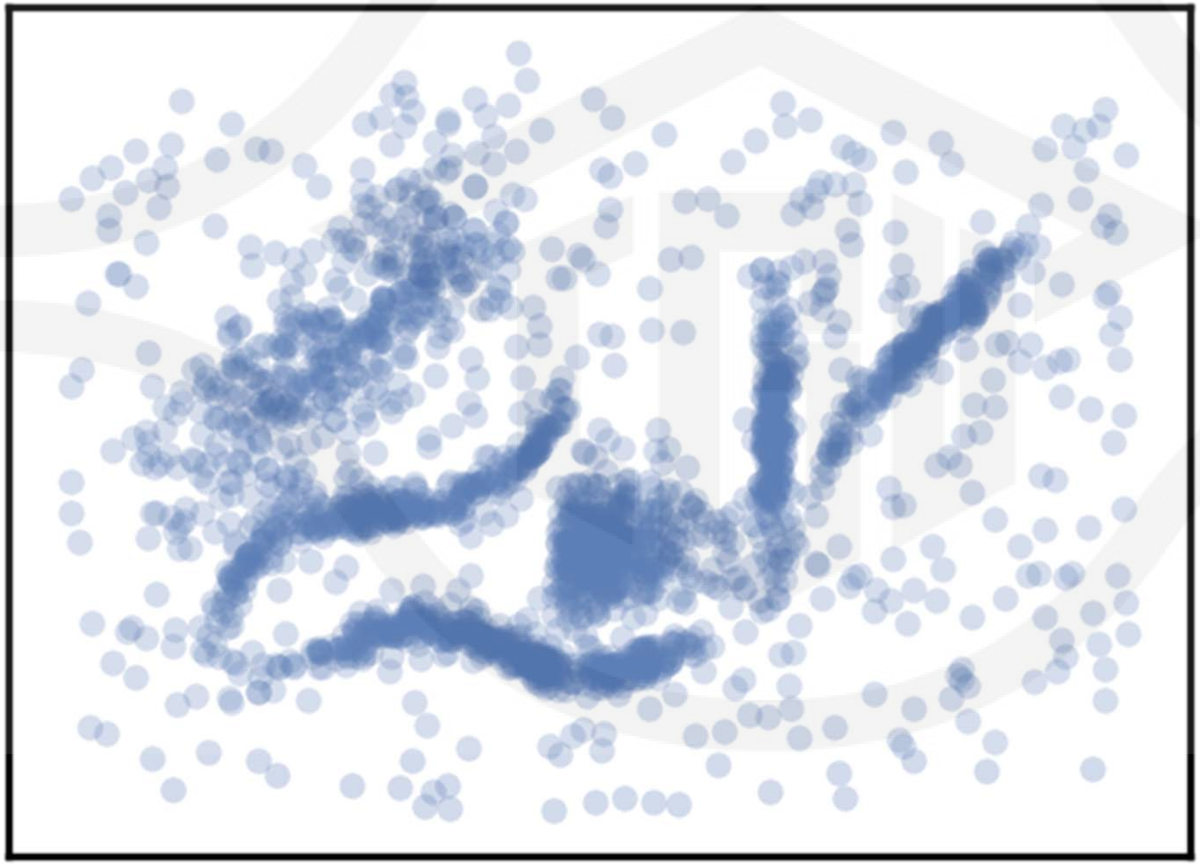
```
from sklearn.cluster import DBSCAN
db = DBSCAN(eps=0.2, min_samples=5, metric='euclidean')
y_db = db.fit_predict(X)
```





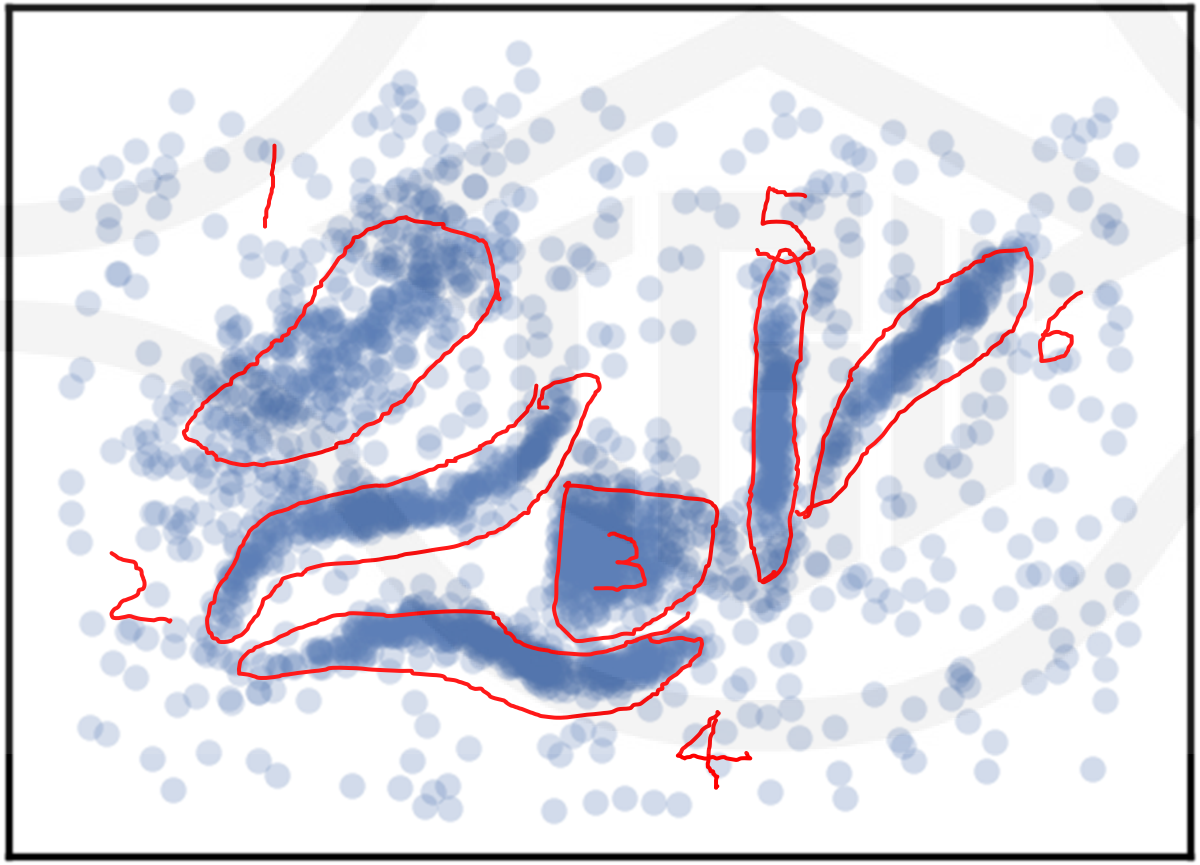
Clustering Experiment

- Cluster this data



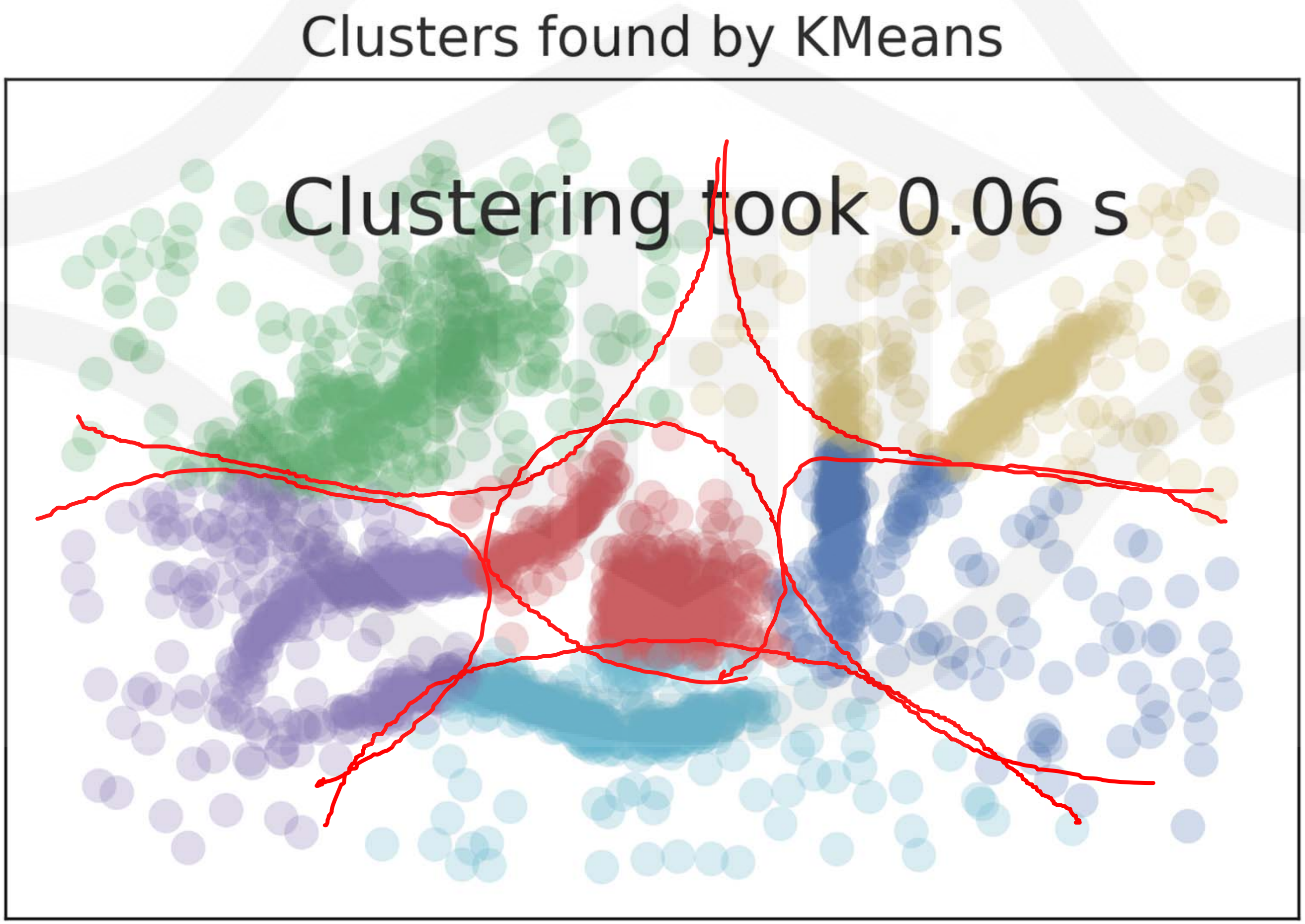
Select k=6

- It seems that there are six groups



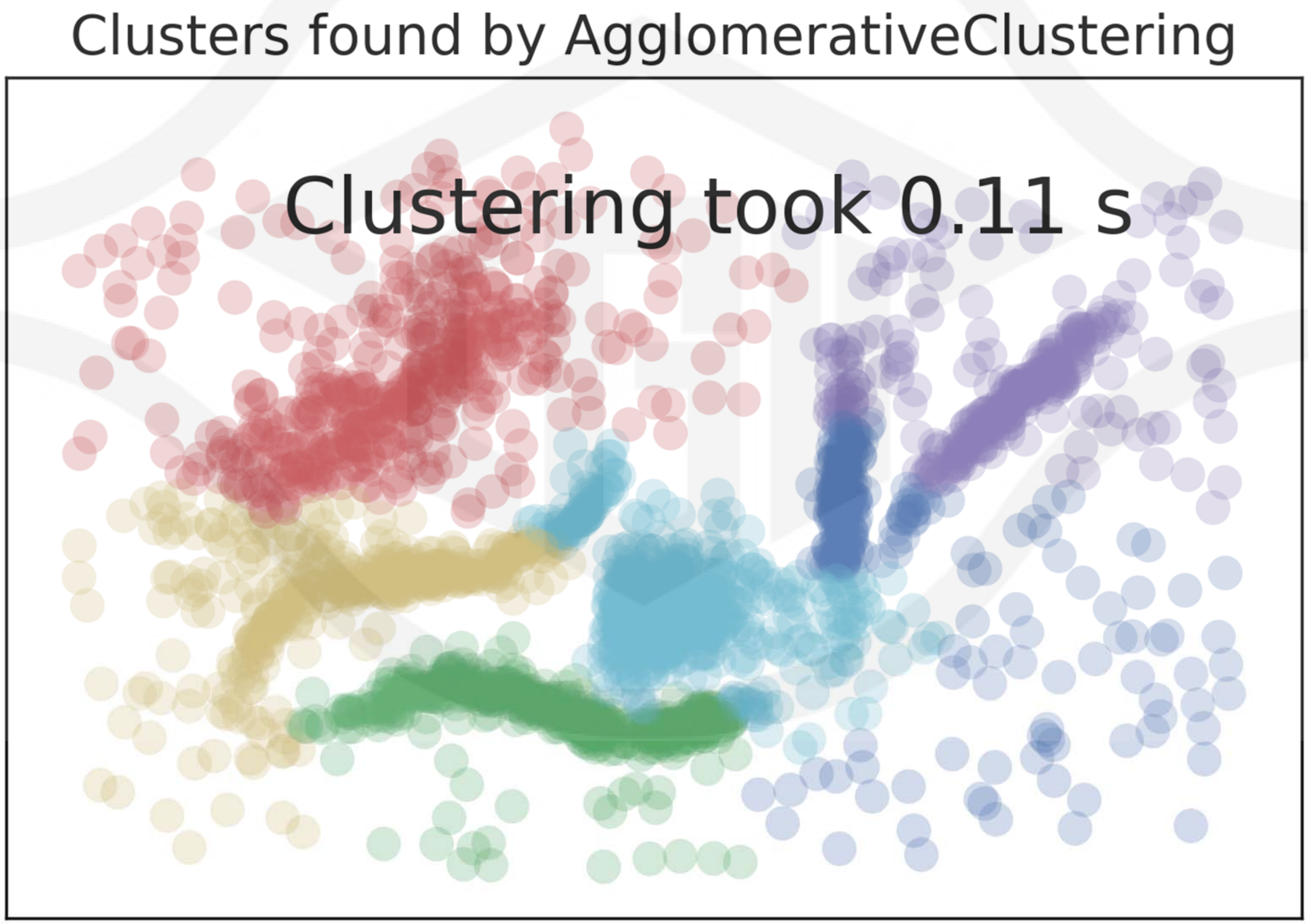
K-mean

- Normal、Spherical shape



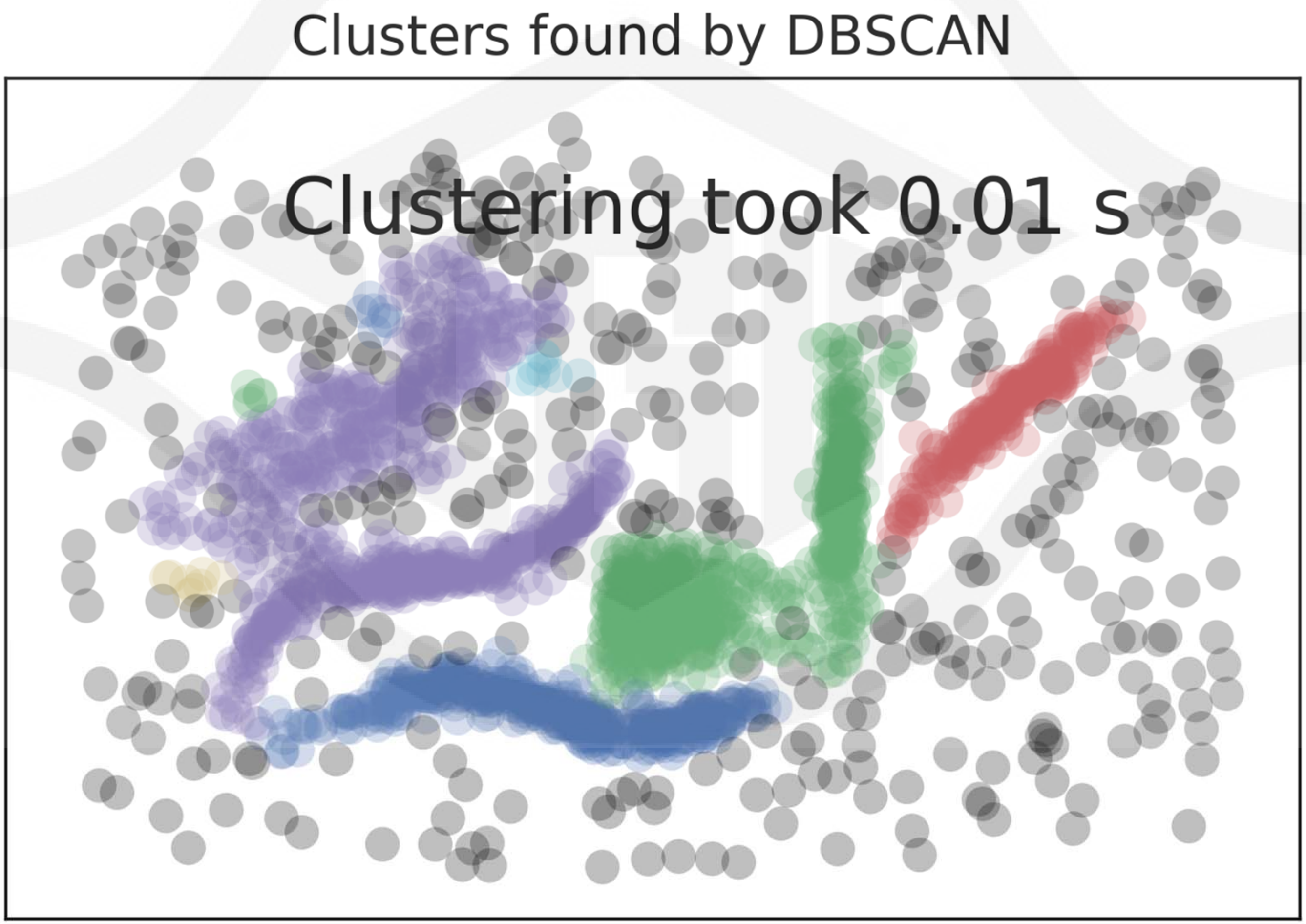
Agglomerative

- Slow、Close to ideal clustering



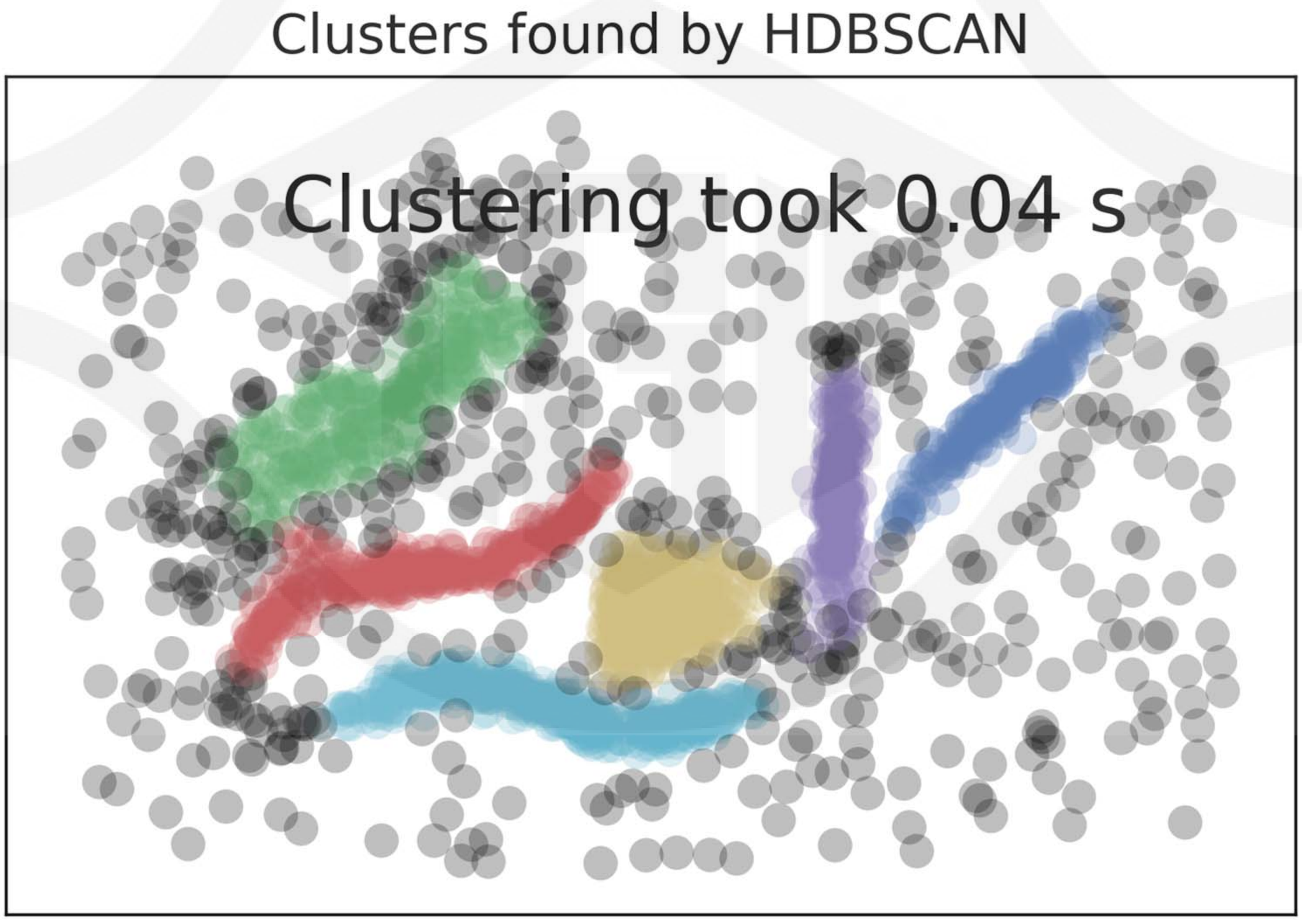
DBSCAN

- Very fast



H-DBSCAN (optional package) not in sitkit-learn

- Fast



Reference

- Sebastian Raschka, Vahid Mirjalili. Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow. Second Edition. Packt Publishing, 2017.
- How HDBSCAN Works https://hdbscan.readthedocs.io/en/latest/how_hdbscan_works.html