Cluster analysis (2)

Dr. Huiping Cao

Hierarchical Methods

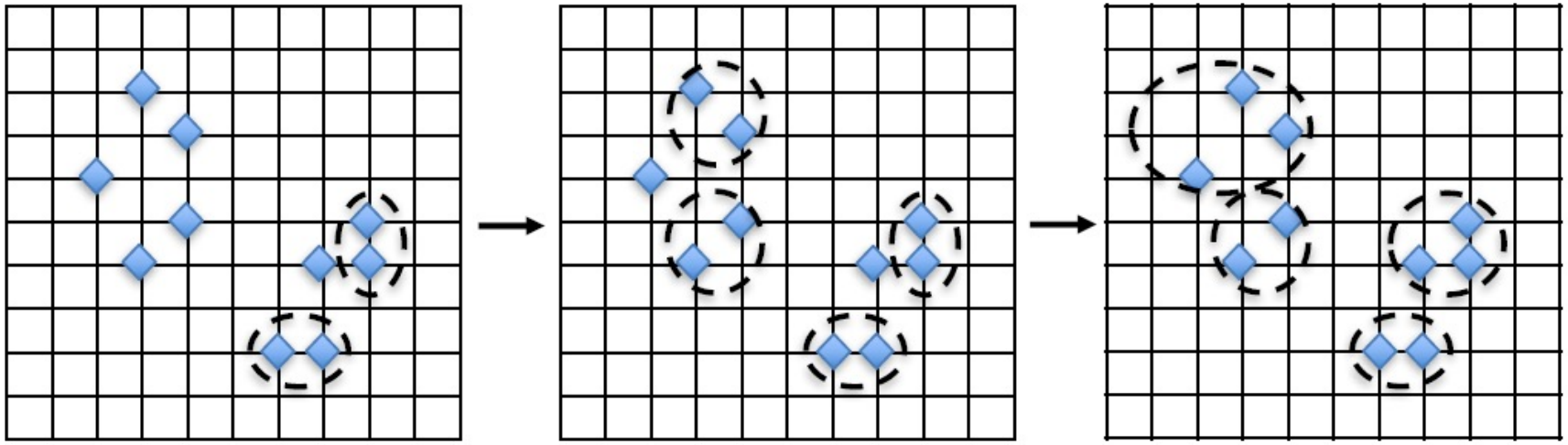
- Use distance matrix as clustering criteria. This method does not require the number of clusters *k* as an input, but needs a **termination condition**.
- **Agglomerative** hierarchical clustering; bottom-up
- **Divisive** hierarchical clustering; top-down

- **Advantages**
  - (1) do not need to specify the number of clusters, and
  - (2) can use dendrograms to visualize the possible results.

AGNES (Agglomerative Nesting)

- Introduced in Kaufmann and Rousseeuw (1990)
- Algorithm
  - Compute the distance matrix of all samples.
  - **Start** by letting each object form its own clusters
  - **Iteratively merge** clusters
    - Merge **two clusters that are closest** to each other
    - Update the similarity matrix
  - **Termination**: all nodes belong to the same cluster
- **Analysis**: Each iteration merges two clusters. This method requires at most *n* iterations.

AGNES - example



Python code to conduct clustering

- Step 1: prepare data
  - 1.1 Generate random points

```
import numpy as np

np.random.seed(123)
X = np.random.random_sample([5,2]) #generate 5*2 random samples which are in the range of [0,1]
X = X*10 #make the values to be in the range of [1 ,10]
print(X)
```

```
[[6.96469186 2.86139335]
 [2.26851454 5.51314769]
 [7.1946897 4.2310646 ]
 [9.80764198 6.84829739]
 [4.80931901 3.92117518]]
```

Python code to conduct clustering

- Step 1: prepare data
  - 1.2 Reformat this to a data frame

```
import pandas as pd

features = ['f1', 'f2']
row_labels = ['p0', 'p1', 'p2', 'p3', 'p4']
df=pd.DataFrame(X,columns=features, index=row_labels)
print(df)
```

```
        f1          f2
p0      6.964692    2.861393
p1      2.268515    5.513148
p2      7.194690    4.231065
p3      9.807642    6.848297
p4      4.809319    3.921175
```

Python code to conduct clustering

- Step 1: prepare data
  - 1.3 (OPTIONAL) Use SciPy's submodule to calculate pair-wise point distance

```
### Calculate pair−wise point distance
from scipy.spatial.distance import pdist,squareform
from scipy.spatial import distance

pair_wise_dist_condensed_form = pdist(df, metric='euclidean')
print(pair_wise_dist_condensed_form)
row_dist = pd.DataFrame(squareform(pair_wise_dist_condensed_form),
                        columns=row_labels, index= row_labels)
print(row_dist)
```

```
[5.3931329 1.38884785 4.89671004 2.40182631
5.09027885 7.6564396 2.99834352 3.69830057
2.40541571 5.79234641]
```

	p0	p1	p2	p3	p4
p0	0.000000	5.393133	1.388848	4.896710	2.401826
p1	5.393133	0.000000	5.090279	7.656440	2.998344
p2	1.388848	5.090279	0.000000	3.698301	2.405416
p3	4.896710	7.656440	3.698301	0.000000	5.792346
p4	2.401826	2.998344	2.405416	5.792346	0.000000

SciPy’s pdist

- **pdist**: scipy.spatial.distance.pdist(X, metric=’euclidean’, *args, **kwargs): Pairwise distances between observations in n-dimensional space.
  - **X**: ndarray : An m by n array of m original observations in an n-dimensional space.
  - **metric**: str or function, optional. The distance metric to use. The distance function can be ’braycurtis’, ’canberra’, ’chebyshev, **’cityblock**’, ’correlation’, ’cosine’, ’dice’, b’, ’hamming’, ’jaccard’, ’kulsinski’, ’mahalanobis’, ’matching’, ’minkowski’, ’rogerstanimoto’, ’russellrao’, ’seuclidean’, ’sokalmichener’, ’sokalsneath’, ’sqeuclidean’, ’yule’.
  - **Returns** a condensed distance matrix *Y*.
- **Squareform:** scipy.spatial.distance.squareform(X, force=’no’, checks=True): converts between condensed distance matrices and square distance matrices.
  - Convert a vector-form distance vector to a square-form distance matrix, and vice-versa.
  - **Returns**: If a condensed distance matrix is passed, a redundant one is returned, or if a redundant one is passed, a condensed distance matrix is returned.

Step 2: Cluster using SciPy function *linkage*

- Apply **scipy.cluster.hierarchy.linkage** to perform hierarchical or agglomerative clustering.
- **scipy.cluster.hierarchy.linkage(y, method='single', metric='euclidean', optimal_ordering=False)**
- **y**: The input y may be either a 1d condensed distance matrix or a 2d array of observation vectors. E.g., in the above example, you can either pass **pair_wise_dist_condensed_form** or directly pass the data **df.value**
- **Return**: A (n − 1) by 4 matrix Z. At the i-th iteration, clusters with indices $Z[i, 0]$ and $Z[i, 1]$ are combined to form cluster $n + i$.
  - A cluster with an index less than n corresponds to one of the n original observations.
  - The distance between clusters $Z[i, 0]$ and $Z[i, 1]$ is given by $Z[i, 2]$.
  - The fourth value $Z[i, 3]$ represents the number of original observations in the newly formed cluster.
- **Metric:** The distance metric to use in the case that y is a collection of observation vectors; ignored otherwise.
  - See the **pdist** function for a list of valid distance metrics. A custom distance function can also be used

Step 2: Cluster using SciPy function *linkage*

```
from scipy.cluster.hierarchy import linkage

row_cluster1 = linkage(df.values, method='complete',
                       metric='euclidean')
print(row_cluster1)
```

- The above code is equivalent to the following code.

```
row_cluster2 = linkage(pair_wise_dist_condensed_form, method='complete')
print(row_cluster2)
```

```
[[0.         2.         1.38884785 2. ]
 [4.         5.         2.40541571 3. ]
 [1.         6.         5.3931329  4. ]
 [3.         7.         7.6564396  5. ]]
```

Explanations

	p0	p1	p2	p3	p4
p0	0.000000	5.393133	1.388848	4.896710	2.401826
p1	5.393133	0.000000	5.090279	7.656440	2.998344
p2	1.388848	5.090279	0.000000	3.698301	2.405416
p3	4.896710	7.656440	3.698301	0.000000	5.792346
p4	2.401826	2.998344	2.405416	5.792346	0.000000

- Clusters C0 (p0) and C2 (p2) are combined to form a cluster (C5).
  - The distance is max(d(p0,p2))=1.388848.
  - C5 contains 2 points (p0, p2).
- Clusters C4 (p4) and C5 (p0, p2) are combined to form a cluster (C6).
  - The distance is *max*(*d*(*p*4, *p*0), *d*(*p*4, *p*2)) = max(2.401826, 2.405416)=2.405416
  - C6 contains 3 points (p0, p2, p4).
- Clusters C1 (p1) and C6 (p0, p2, p4) are combined to form a cluster (C7).
  - The distance is max(d(p1,p0), d(p1, p2), d(p1, p4)) = max(5.393133, 5.090279, 2.998344) = 5.393133
  - C7 contains 4 points (p0, p2, p4, p1)
- Clusters C3 (p3) and C7 (p0, p2, p4, p1) are combined to form a cluster (C8).
  - The distance is max(d(p3,p0), d(p3, p2), d(p3, p4), d(p3, p1)) = max(4.896710, 3.698301, 5.792346,7.656440) = 7.656440.
  - C8 contains all the five points.

```
[[0.         2.         1.38884785 2. ]
 [4.         5.         2.40541571 3. ]
 [1.         6.         5.3931329  4. ]
 [3.         7.         7.6564396  5. ]]
```

Step 2: Cluster using SciPy function *linkage*

- **Algorithm**:
  - **Several linkage methods** are used to compute the distance $d(s, t)$ between two clusters s and t.
  - The algorithm **begins with a forest of clusters** that have yet to be used in the hierarchy being formed.
  - When **two clusters** s and t from this forest **are combined into a single cluster** u, s and t are removed from the forest, and u is added to the forest.
  - When **only one cluster remains** in the forest, the algorithm **stops**, and this cluster becomes the root.
  - A **distance matr**ix is maintained at each iteration. The $d[i, j]$ entry corresponds to the distance between cluster i and j in the original forest.
  - At **each iteration**, the algorithm must update the distance matrix to reflect the distance of the newly formed cluster u with the remaining clusters in the forest.

Step 2: Cluster using SciPy function *linkage*

- **Linkage methods**: 'single', 'complete', 'average', 'weighted', 'centroid', 'median', 'ward'
- **Single linkage**: the smallest distance between an element in one cluster and an element in the other, I.e.,
  - $d(C_i, C_j) = \min_{p,q} d(p,q)$ for $\forall p \in C_i, q \in C_j$
  - Utilizing this method is also known the Nearest Point Algorithm.
- **Complete link**: the largest distance between an element in one cluster and an element in the other, i.e.,
  - $d(C_i, C_j) = \max_{p,q} d(p,q)$ for $\forall p \in C_i, q \in C_j$
  - This is also known by the Farthest Point Algorithm or Voor Hees Algorithm.
- **Average**: the average distance between an element in one cluster and an element in the other, i.e.,
  - $d(C_i, C_j) = \sum_{p,q} \frac{d(p,q)}{|C_i| \times |C_j|}$ for $\forall p \in C_i, q \in C_j$ and $|C_i|$ is the number of points in cluster C_i.
  - This is also called the UPGMA algorithm.
- **Weighted**: $d(C_i, C_j) = (d(C_{i1}, C_j) + d(C_{i2}, C_j))/2$
  - cluster C_i was formed with cluster C_{i1} and C_{i2}, and C_j is a remaining cluster in the forest. (also called WPGMA)

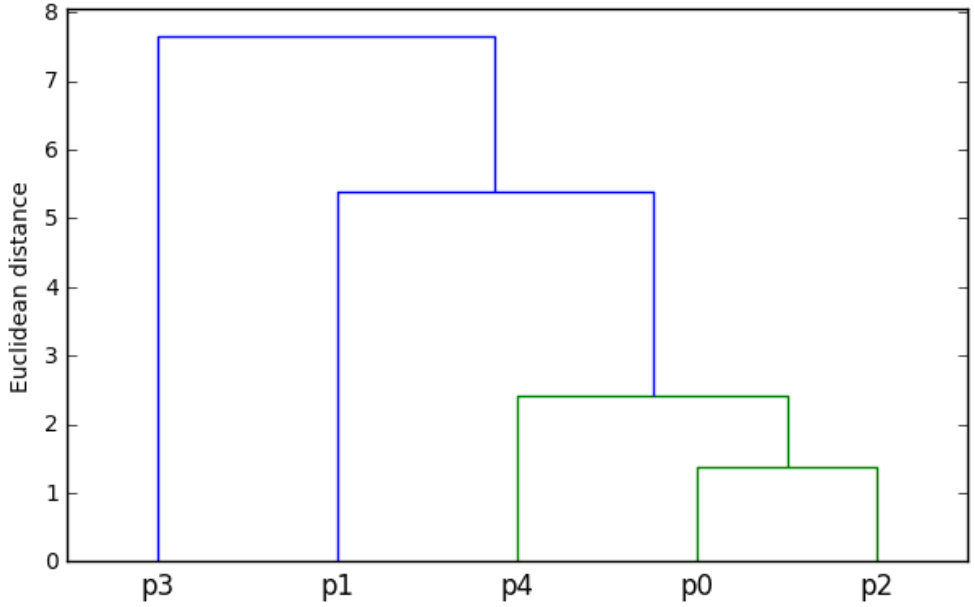
Step 2: Cluster using SciPy function *linkage*

- **Centroid**: $d(C_i, C_j) = d(c_i, c_j)$ where c_i and c_j are the centroids of the two clusters C_i and C_j. The new centroid is computed over all the original objects in clusters C_i and C_j.
  - This is also known as the UPGMC algorithm.
- **Median:** It assigns $d(C_i, C_j)$ like the centroid method. When two clusters C_i and C_j are combined into a new cluster, the average of centroids C_i and C_j give the new centroid. This is also known as the WPGMC algorithm.
- **Ward's linkage**: It uses the Ward variance minimization algorithm. The new entry *d*(*u*,*v*) is computed as follows,
  - $d(u,v) = \sqrt{\frac{|v|+|C_i|}{T} d(v,C_i)^2 + \frac{|v|+|C_j|}{T} d(v,C_j)^2 - \frac{|v|}{T} d(C_i,C_j)^2}$
  - where *u* is the newly joined cluster consisting of clusters C_i and C_j, *v* is an unused cluster in the forest, $T = |v| + |C_i| + |C_j|$ and $|*|$ is the cardinality of its argument.
  - This is also known as the incremental algorithm.
  - Intuitively, it merges two clusters that lead to the minimum increase of the total within-cluster SSE are merged.

Step 3: Dendrogram: Shows How the Clusters are Merged

- Decompose data objects into several levels of nested partitioning (tree of clusters), called a **dendrogram**.
- A clustering of the data objects is obtained by cutting the dendrogram at the desired level, then each connected component forms a cluster.

```
from scipy.cluster.hierarchy import dendrogram

row_dendr = dendrogram(row_cluster1, labels=row_labels)

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



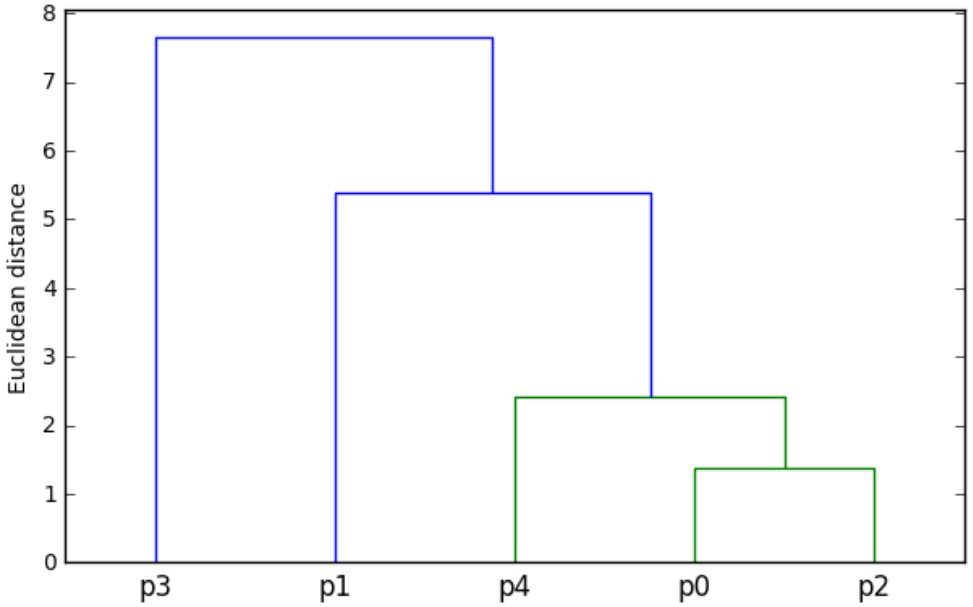
Step 4: retrieve the final clusters using fcluster

```
from scipy.cluster.hierarchy import fcluster

max_d = 2.5
clusters1 = fcluster(row_cluster1, max_d, criterion='distance')
print(clusters1)

k=2
clusters2 = fcluster(row_cluster1, k, criterion='maxclust')
print(clusters2)
```

```
[1 2 1 3 1]

[1 1 1 2 1]
```

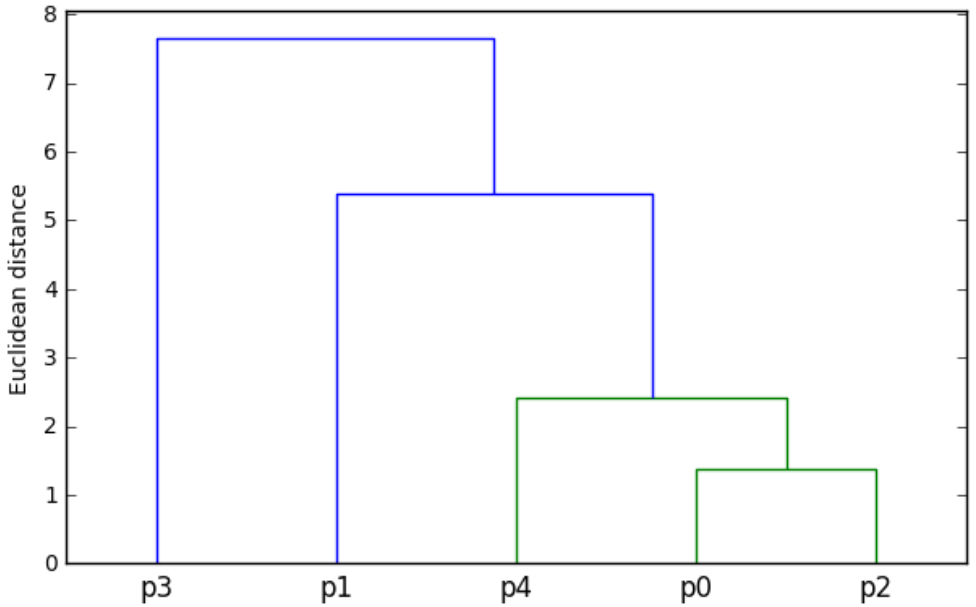
- scipy.cluster.hierarchy.fcluster(Z, t, criterion='inconsistent', depth=2, R=None, monocrit=None): form flat clusters from the hierarchical clustering defined by the linkage matrix Z.
  - Z: The hierarchical clustering encoded with the matrix returned by the linkage function.
  - t: The threshold to apply when forming flat clusters.
  - criterion:
    - distance: Forms flat clusters so that the original observations in each flat cluster have no greater cophenetic distance than t.
    - maxclust: Finds a minimum threshold *r* so that (1) the cophenetic distance between any two original observations in the same flat cluster is no more than *r* and (2)s no more than t flat clusters are formed.
- The **cophenetic distance** between two objects is the height of the dendrogram where the two branches that include the two objects merge into a single branch. Outside the context of a dendrogram, it is the distance between the largest two clusters that contain the two objects individually when they are merged into a single cluster that contains both.

Explanation of the results



3 2 1 1 1

max_d=2.5
[1 2 1 3 1] c



2 1 1 1 1

k=2
[1 1 1 2 1]

Cluster using scikit learn's *AgglomerativeClustering*

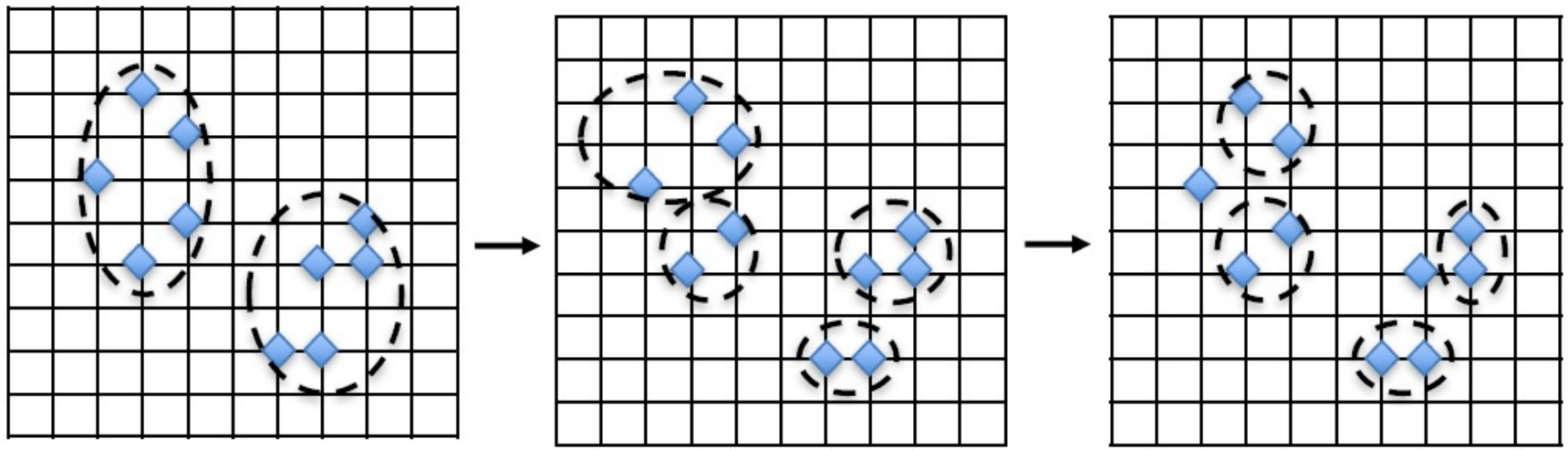
```
from sklearn.cluster import AgglomerativeClustering

cluster1 = AgglomerativeClustering(n_clusters=3,          affinity='euclidean', linkage='complete')

cluster1_labels = cluster1.fit_predict(X)
print(cluster1_labels)
```

```
[0 2 0 1 0]
```

- Cluster 0: p0, p2, p4
- Cluster 1: p3
- Cluster 2: p1

DIANA (Divisive Analysis)

- Introduced in Kaufmann and Rousseeuw (1990)
- **Start** by placing all objects in one cluster
- **Iteratively divides** clusters
- **Eventually** (1) each node forms a cluster on its own, or (2) objects within a cluster are sufficiently similar to each other.



DIANA (Divisive Analysis)

- Inverse order of AGNES
- Practically, use heuristics in partitioning because there are $2^{n-1} - 1$ possible ways to partition a set of *n* objects into two exclusive subsets.
- There are many more agglomerative methods than divisive methods.

Hierarchical Clustering Methods: discussions

- Major weakness of agglomerative clustering methods
  - Do not scale well: time complexity of at least $O(n^2)$, where n is the number of total objects
- Further improvement
  - BIRCH (1996): uses CF-tree and incrementally adjusts the quality of sub-clusters
  - ROCK (1999): clustering categorical data by neighbor and link analysis
  - CHAMELEON (1999): hierarchical clustering using dynamic modeling

Reference

- Chapter 11, Sebastian Raschka and Vahid Mirjalili: Python Machine Learning (Machine learning and deep learning with Python, scikit-learn, and TensorFlow), 3rd Edition.