Module 8

Part 2 – Unsupervised Learning

José Manuel García Nieto – University of Málaga

Table of contents

Supervised versus unsupervised learning

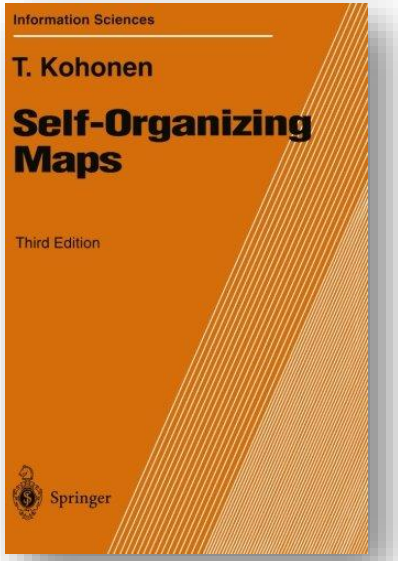
- Supervised learning: classification is seen as supervised learning from examples
  - ❑ Supervision: the data (observations, measurements, etc.) are labeled with pre-defined classes
  - ❑ It is like that a “teacher” gives the classes (supervision)
  - ❑ Test data are also classified into these classes
- Unsupervised learning (clustering)
  - ❑ Class labels of the data are unknown
  - ❑ Given a set of data, the task is to establish the existence of classes or clusters in the data

First, let’s get some intuition about unsupervised learning…

What is unsupervised learning and cluster analysis?

"[Unsupervised learning is] learning without *a priori* knowledge about the classification of samples; learning without a teacher"

Kohonen, "Self-Organizing Maps"



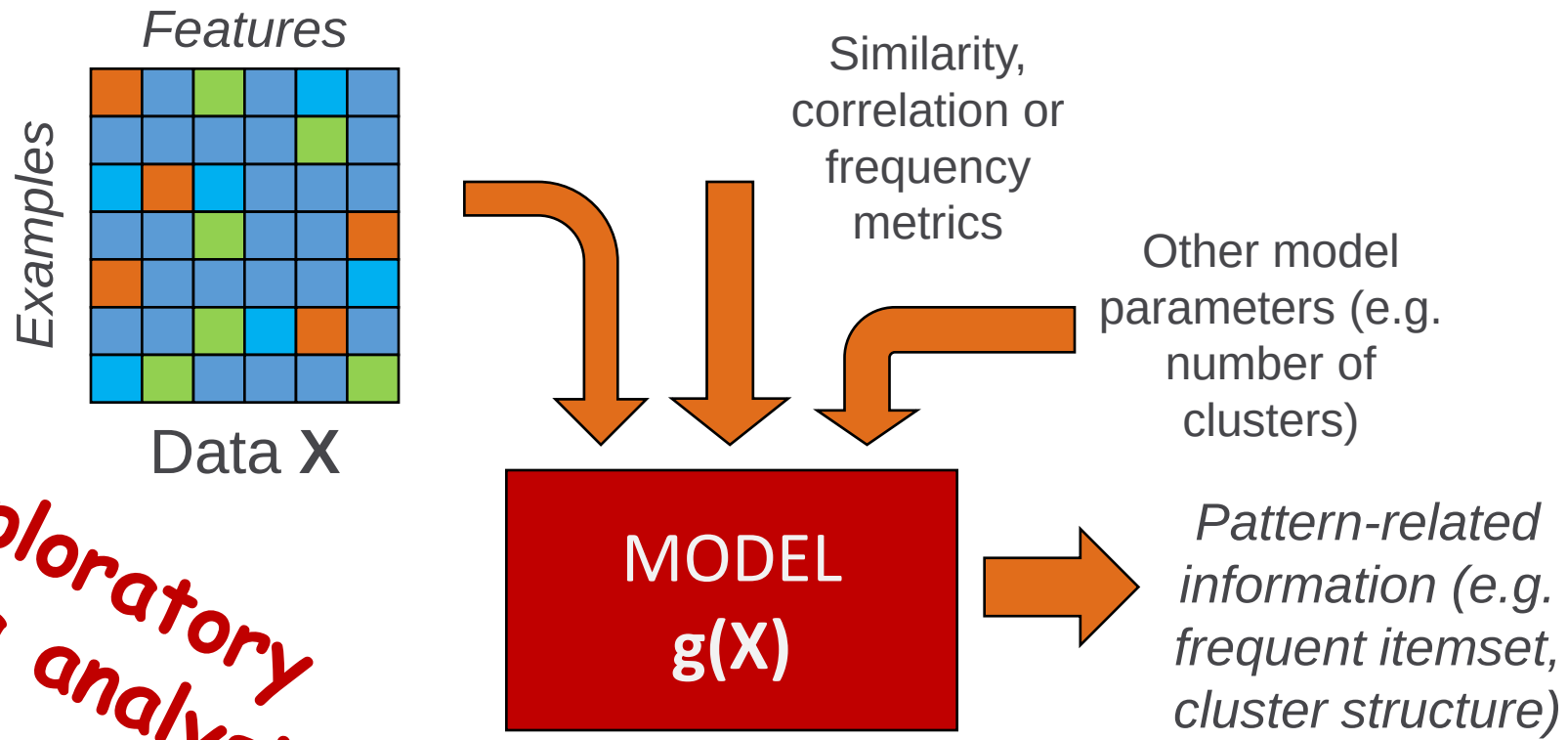
It is a machine learning technique performed only when a series of variables such as $x_1, x_2, x_3, \ldots x_p$ is given without any objective variables (or response variable Y) in a data set.

Since no objective variables (or response variables) are associated with explanatory variables, unsupervised learning aims **to discover a specific pattern or unknown knowledge** from given data **instead of making a prediction**.

"Cluster analysis is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense or another) to each other than to those in other groups (clusters)"

NO LABELS! Unsupervised learning

- Scoring function cannot be defined as a function of the data, but rather in terms of its interpretability (pattern discovery)
- Scoring evolves to similarity, correlation or frequency metrics



Machine learning for unsupervised learning

- **Clustering** refers to making groups of similar objects or people according to the analysis purpose and algorithm.
- **Association** refers to uncovering a co-occurrence relationship between specific items in a dataset.
- **Dimensionality reduction** refers to summarizing a given variable set into a smaller number of variables for providing effective explanations.

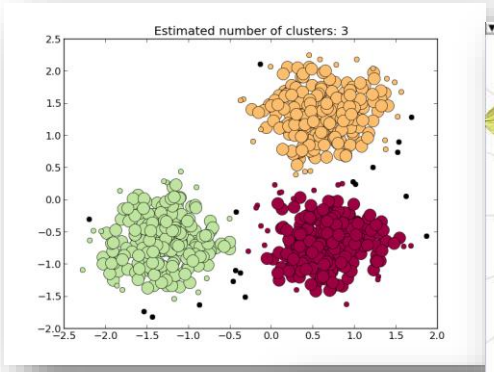
So, in summary…What is cluster analysis used for?

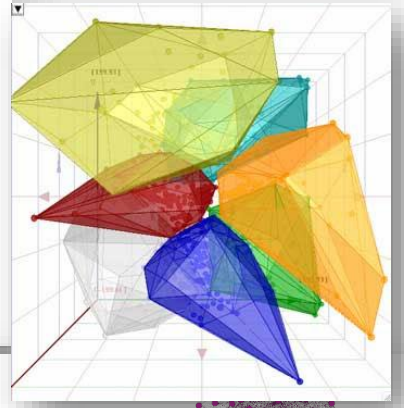
1) Data Exploration: finding patterns, correlations and similarities between multi-dimensional input data

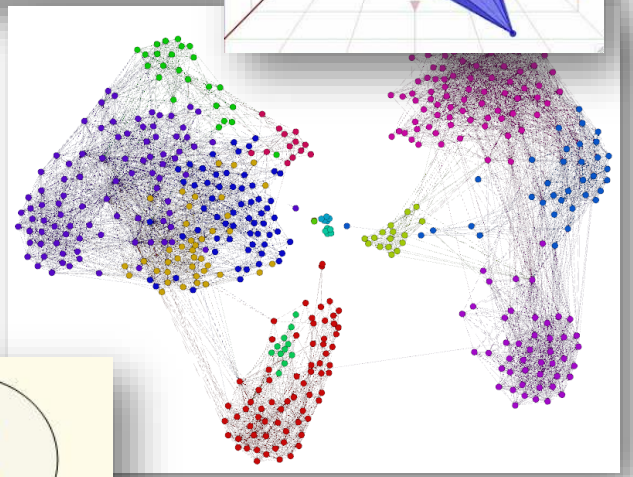
2) Data summarization: a preprocessing step for reducing the size of large data sets

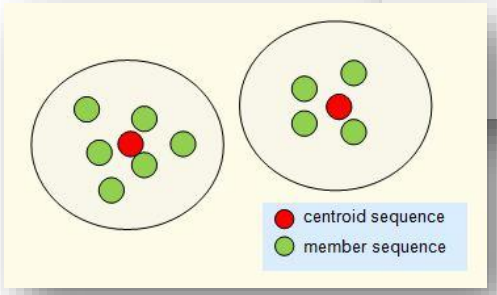
- Represent “clouds” of data instances by their most representative point (“centroids”)

Clustering is one of the **most utilized** data mining techniques









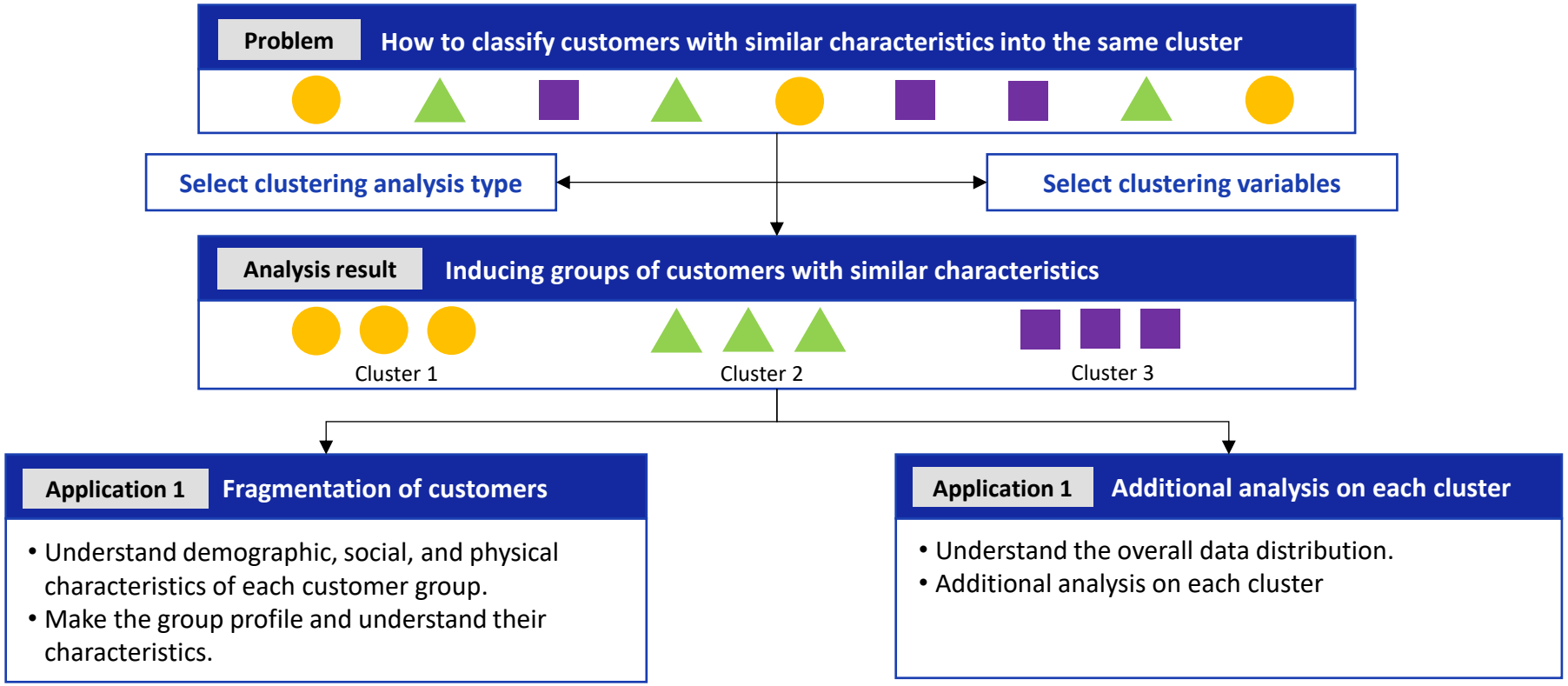
Cluster analysis and unsupervised learning

Some real life examples…

- A marketing designer: “How do customers behave in terms of purchases?
- Could I exploit this information in targeted marketing campaigns? Can we offer our clients products that other customers similar to them already own and they don’t?
- A motor insurance officer: “Are there policy holders with certain similar characteristics (type of car, color, age, etc) whose claims fall within a certain cost range?”
- A financial agent: “Could I group stocks with similar price fluctuations?”
- An energy dealer: “Could I discover distinct energy consumption patterns of a certain user towards personalized contracts or detection of NTL?
- A bioinformatics engineer: “It would be wonderful to group genes and proteins that have similar functionality”
- Identification of areas of similar use from DB with observations of the land (crops, ...)
- Urban planning: Identify groups of houses according to their type, value or geographical location
- Clustering documents, music, or movies into different themes

What is clustering analysis?

Clustering is a technique that divides unlabeled and uncategorized data into similar groups based on the given observed values.

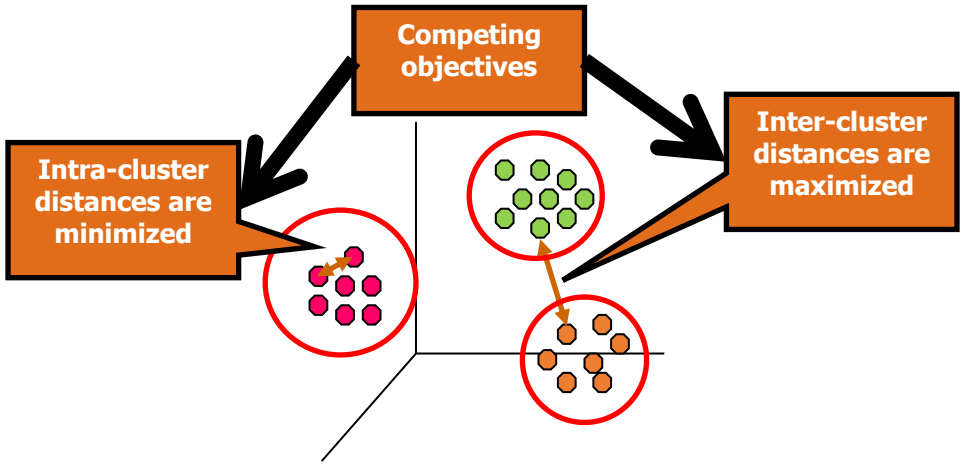


How is a clustering analysis defined?

1. A clustering algorithm and its parameters
   - Partitional clustering
   - Hierarchical clustering
   - Model based clustering
   - Density-based clustering
   - Spectral clustering
   - Others (e.g. meta-heuristic clustering)

2. A distance (similarity, or dissimilarity) function

3. A criterion for clustering quality measurement
   - Inter-clusters distance ⇒ maximized
   - Intra-clusters distance ⇒ minimized
   - Other? (application-dependent)



The quality of a clustering result depends on the algorithm, the distance function and the application.

Some prerrequisites

Cluster definitional ambiguity and similarity metric

No landmark definition of “cluster”: when to stop?

Let them be posed by the nature of the application scenario being tackled

Four Clusters

Six Clusters

The chosen similarity metric (e.g. Euclidean distance or Pearson correlation) impacts on the recovered clusters

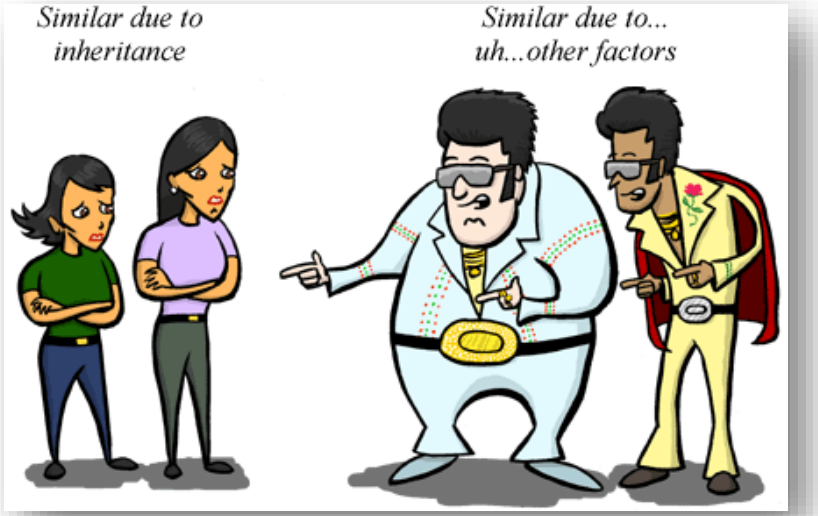
Similarity? Distance?

- Key to clustering: “distance” and “dissimilarity” are also commonly used terms in the clustering community

- Similarity can be understood as a function that determines to which extent two examples (samples, individuals, instances) of the dataset to be clustered differ from each other

- There are numerous distance functions for
  - Different types of features
    - Numeric features
    - Nominal features
  - Different specific applications

(c) Eamonn Keogh, eamonn@cs.ucr.edu

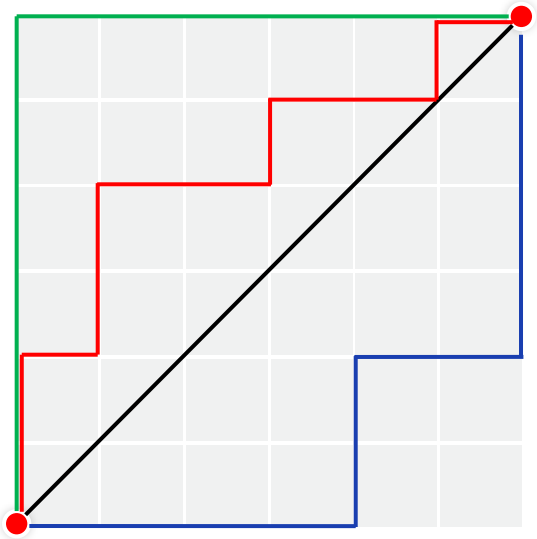


Similarity = f (context)!!!

Typical measures of similarity

They also depend on the kind of attributes

- The definition of the distance measure usually depends on the type of variable:
  - Continuous variables
  - Binary / Boolean variables
  - Nominal / categorical variables
  - Ordinal variables
  - Mixed variables



- The simplest case (a single numeric attribute):
- Various numeric attributes: Minkowski distance

$$d(i,j) = |x_{i1} - x_{j1}|$$

$$d(i,j) = \sqrt[q]{(|x_{i1} - x_{j1}|^q + |x_{i2} - x_{j2}|^q + \ldots + |x_{ip} - x_{jp}|^q)}$$

$q = 2$: Euclidean distance

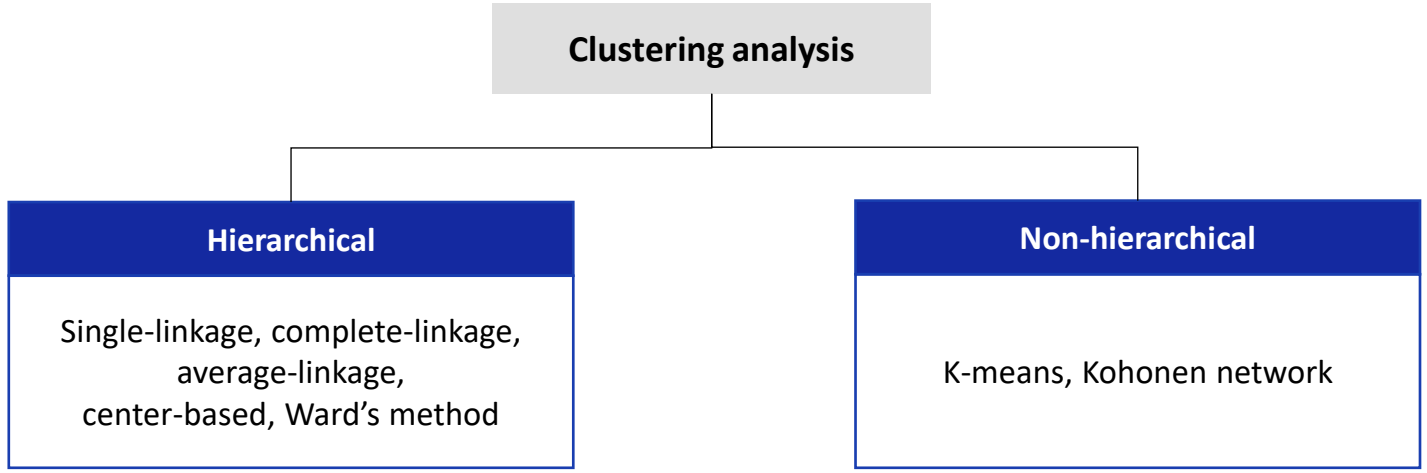
$q = 1$: Manhattan distance

- Nominal attributes: Distance is set to 1 if values are different, 0 if they are equal
- Many measures are sensitive to the range of each variable, so it is necessary to normalize them
- Are all the attributes of equal importance?
- If they are not of equal importance, it will be necessary to weight attributes:

$$d(i,j) = \sqrt{w_1|x_{i1} - x_{j1}|^2 + w_2|x_{i2} - x_{j2}|^2 + \ldots + w_p|x_{ip} - x_{jp}|^2}$$

Two big types of clustering analysis

- Hierarchical clustering: divisive and agglomerative
- Non-hierarchical clustering: the number of clusters is determined in advance



Two big types of clustering analysis

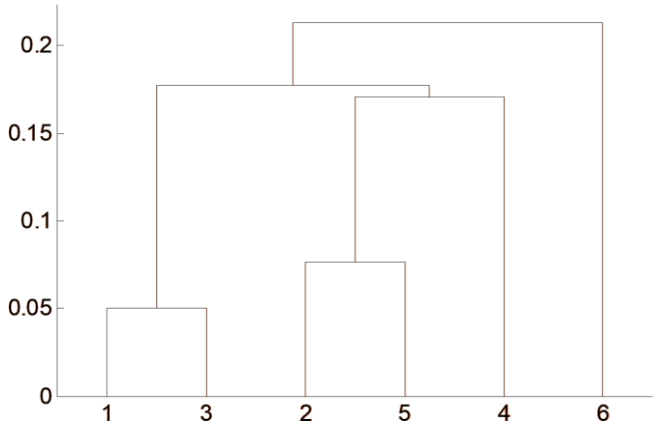
- In general, clustering analysis is classified into hierarchical clustering, non-hierarchical clustering, and partition-based clustering.

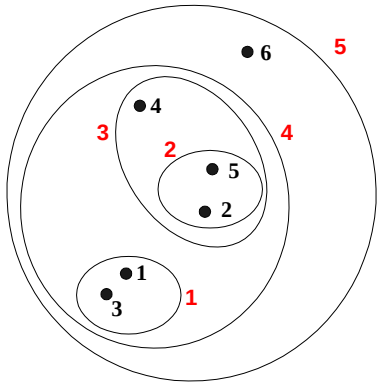
Non-Hierarchical Clustering (Partition-based Clustering)	K-means clustering
	K-medoids clustering or PAM (Partitioning Around Method)
	DBSCAN (Density Based Spatial Clustering of Application with Noise)
	Self Organizing Map
	Fuzzy clustering
Hierarchical Clustering	Agglomerative or bottom-up clustering
	Divisive or top-down clustering
Mixture Distribution Clustering	Gaussian Mixture Model

Table of contents

Hierarchical clustering

- Produces a set of nested clusters arranged as a hierarchical tree
- Can be visualized as a dendrogram
  - A tree diagram that records the sequences of merges or splits





- Does not have to assume any particular number of clusters (advantage)
  - Any desired number of clusters can be obtained by ‘cutting’ the dendogram at the proper level
- They may represent meaningful taxonomies (in context)
  - Example in biological and social sciences (e.g., animal kingdom, phylogeny, reconstruction, hierarchical communities, etc…)

Types of hierarchical clustering

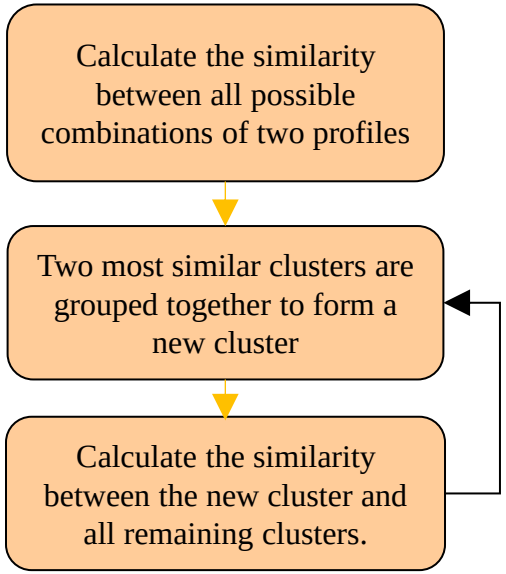
- Agglomerative (bottom up) clustering: it builds the dendrogram (tree) from the bottom level, and
  - merges the most similar (or nearest) pair of clusters
  - stops when all the data points are merged into a single cluster (i.e., the root cluster).

- Divisive (top down) clustering: It starts with all data points in one cluster, the root.
  - Splits the root into a set of child clusters. Each child cluster is recursively divided further
  - stops when only singleton clusters of individual data points remain, i.e., each cluster with only a single point

- Traditional hierarchical algorithms use a similarity or distance matrix
  - Merge or split one cluster at a time

Agglomerative hierarchical clustering

- More often utilized than its divisive counterpart
  - At the beginning, each data point forms a cluster (also called a node).
  - Merge nodes/clusters that have the least distance.
  - Go on merging
  - Eventually all nodes belong to one cluster



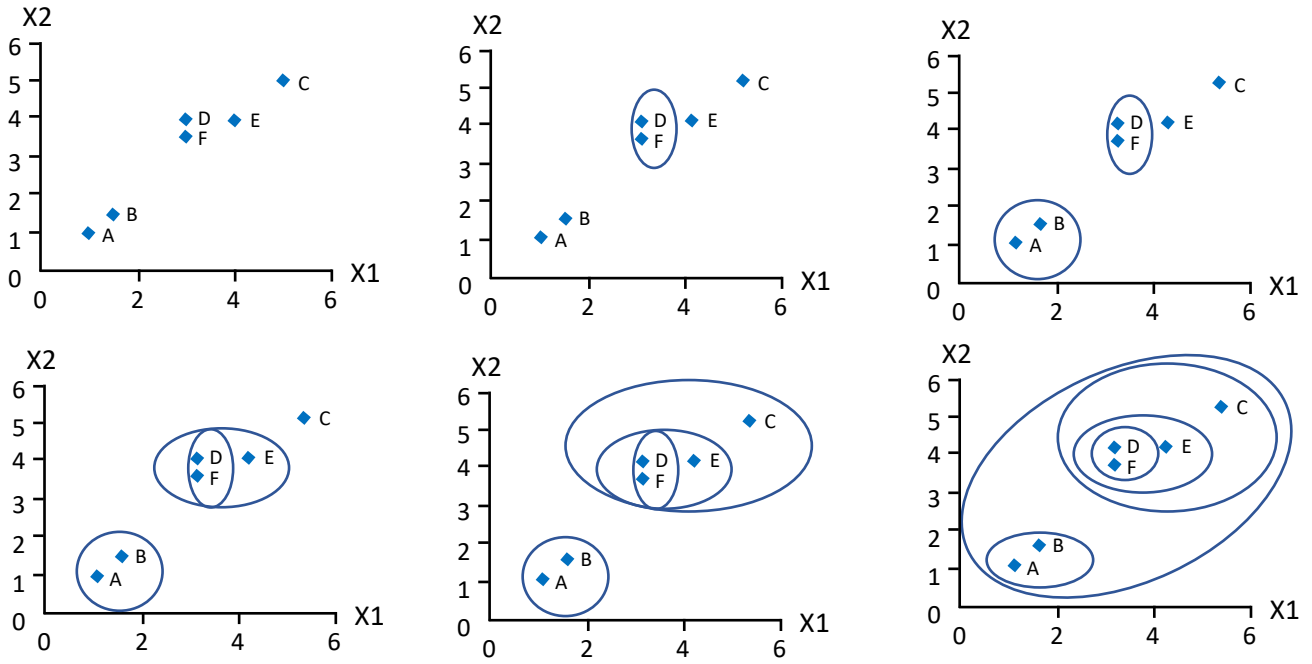
- The Basic algorithm is straightforward

```
1. Compute the proximity matrix
2. Let each data point be a cluster
3. Repeat
4.        Merge the two closest clusters
5.        Update the proximity matrix
6. Until only a single cluster remains
```

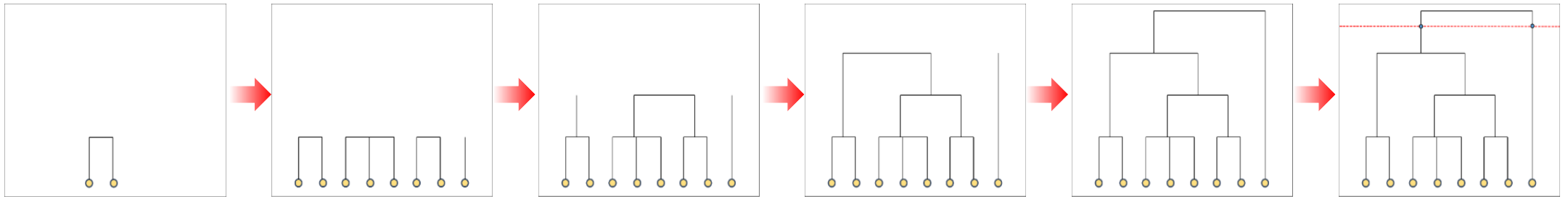
- Key operation is the proximity of two clusters (linkage computation)
  - Different approaches for defining the distance between clusters is what distinguishes among different agglomerative algorithms

Agglomerative hierarchical clustering

- Start: Each object is an individual atomic cluster.
- Repetition: Repeated merging of two closest clusters
- Finish: Generates a single cluster



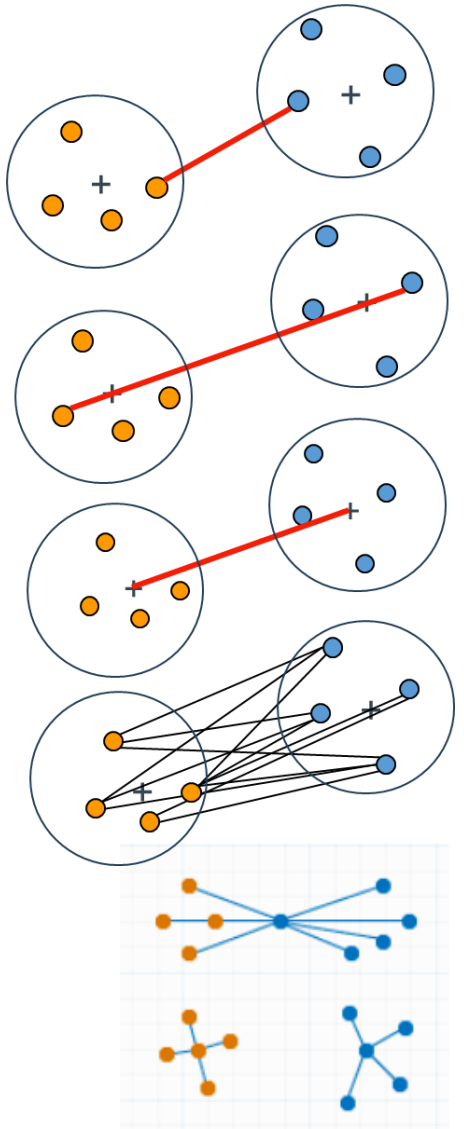
Agglomerative hierarchical clustering



1. First, group together the nearest observations and get small clusters: (a) ~ (b).
2. Group the nearest clusters together to form larger clusters: (c).
3. Repeat step 2) until all the observations are grouped together into a single cluster: (d) ~ (e).
4. Cut the dendrogram at an appropriate height such that we get the desired number of clusters: (f).

Linkage methods

- Single linkage
  - Dissimilarity between two clusters = Minimum dissimilarity between the members of two clusters
  - Tend to form “long chains” (loosely defined chain-shaped clusters due to noise and outliers)
- Complete Linkage
  - Dissimilarity between two clusters = Maximum dissimilarity between the members of two clusters
  - Tend to generate “clumps” (due to the breaking of increasingly globular clusters)
- Average Group Linkage (Centroid)
  - Dissimilarity between two clusters = Distance between two cluster centroids
  - Compromise between Single and Complete Linkage
- Average Linkage
  - Dissimilarity between two clusters = Averaged distances of all pairs of objects (one from each cluster)
- Ward Linkage
  - Determines the distance between two clusters by minimizing the increase in variance when the two clusters are merged. This method tends to produce clusters that have similar variances and sizes.

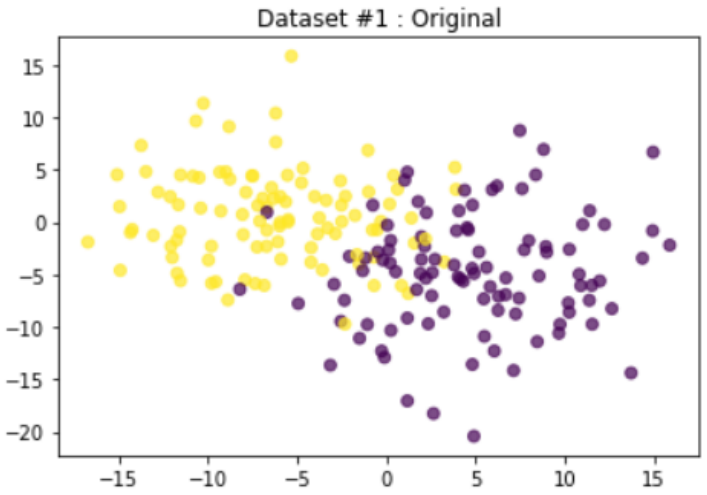


Apply agglomerative clustering and visualize

- Dataset #1: Blobs

In [1]:
```
from sklearn.datasets import make_blobs, make_moons
import matplotlib.pyplot as plt
```

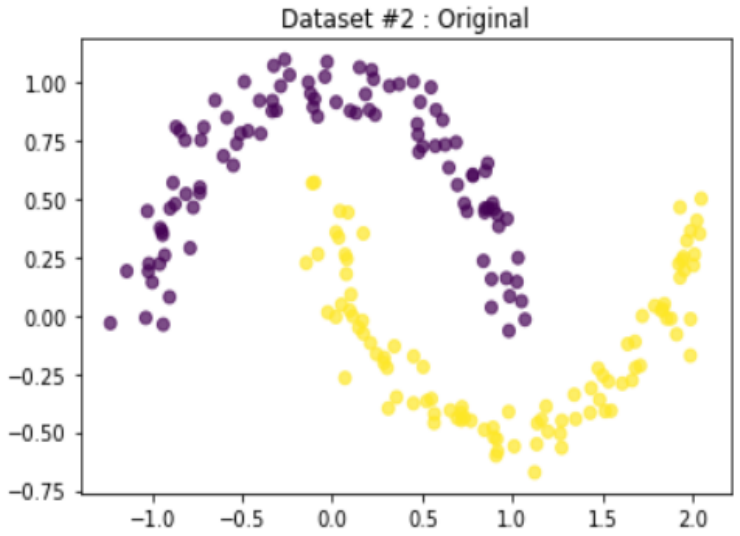
In [2]:
```
X1, label1 = make_blobs(n_samples=200, n_features=2, centers=2, cluster_std = 5, random_state=123)
plt.scatter(X1[:,0],X1[:,1], c= label1, alpha=0.7 )
plt.title('Dataset #1 : Original')
plt.show()
```



Apply agglomerative clustering and visualize

- Dataset #2: Moons

```
In [3]: X2, label2 = make_moons(n_samples=200, noise=0.08, random_state=123)
        plt.scatter(X2[:,0],X2[:,1], c= label2, alpha=0.7 )
        plt.title('Dataset #2 : Original')
        plt.show()
```



Apply agglomerative clustering and visualize

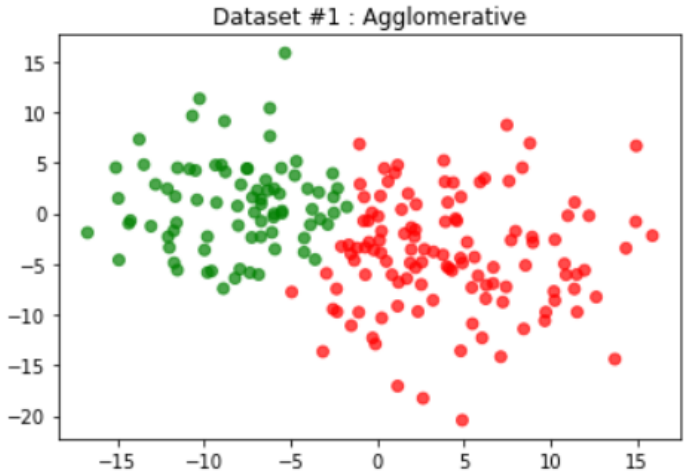
- Clusters

```
In [4]: from sklearn.cluster import AgglomerativeClustering

In [5]: import numpy as np
        import pandas as pd

In [6]: X1

Out[6]: array([[ 4.83455936e+00,  1.61209639e+00],
               [-7.19342110e+00,  1.57331116e+00],
               [-1.19919003e+01, -3.38784956e+00],
               [ 8.94965320e+00, -2.34628131e+00],
               [ 5.46630481e+00, -7.33334000e+00],
               [-9.82679362e-01,  2.87439331e+00],
               [ 4.82713114e+00, -1.35870919e+01],
               [-5.95639441e+00, -4.86198432e-02],
               [ 3.25723451e+00, -3.81242660e+00],
               [-1.49388489e+01,  1.46494770e+00],
               [-9.32588959e+00,  4.75039194e+00],
               [ 1.23752259e+00, -6.81998526e+00],
               [ 1.19728386e+01, -5.62837526e+00],
               [-2.58199902e-01, -1.23070271e+01],
               [ 1.49590341e+01,  6.65671714e+00],
               [ 2.25314845e-01, -3.91267708e+00],
               [-5.93898363e+00,  2.41971297e+00],
               [-4.66024882e+00,  5.12509843e+00],
               [-2.49261965e+00,  3.07334679e-02],
               [ 7.73413068e+00, -2.65986906e+00],
               [-5.12306455e-01, -7.42476287e-01],
               [ 1.57887480e+00, -2.01725863e+00],
               [-4.20565001e+00, -2.44769569e+00],
               [-4.00859899e+00,  2.42460873e+00],
               [ 3.91603180e+00,  3.10476808e+00],
               [-1.05722363e+00, -9.77742886e+00],
```

Apply agglomerative clustering and visualize

- Dataset #1: Blobs and two clusters

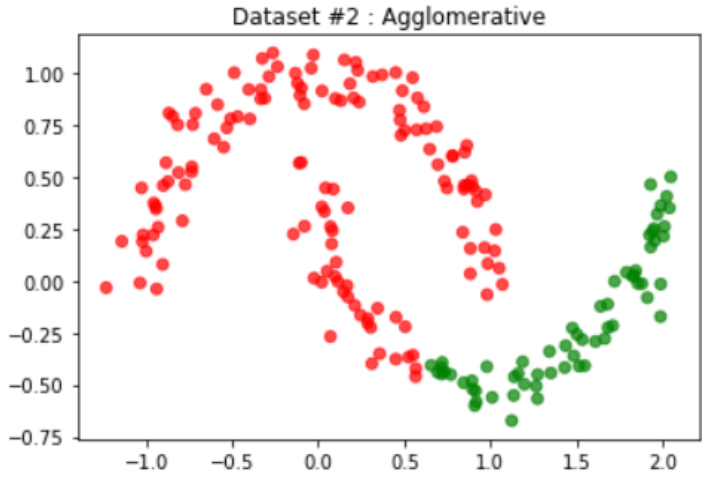
```
In [7]: agglo = AgglomerativeClustering(n_clusters=2)
        agglo.fit(X1)
        myColors = {0:'red',1:'green'}                                  # Define a color palette: 0~1.
        plt.scatter(X1[:,0],X1[:,1], c= pd.Series(agglo.labels_).apply(lambda x: myColors[x]), alpha=0.7 )
        plt.title('Dataset #1 : Agglomerative')
        plt.show()
```



Apply agglomerative clustering and visualize

- Dataset #2: Moons and two clusters

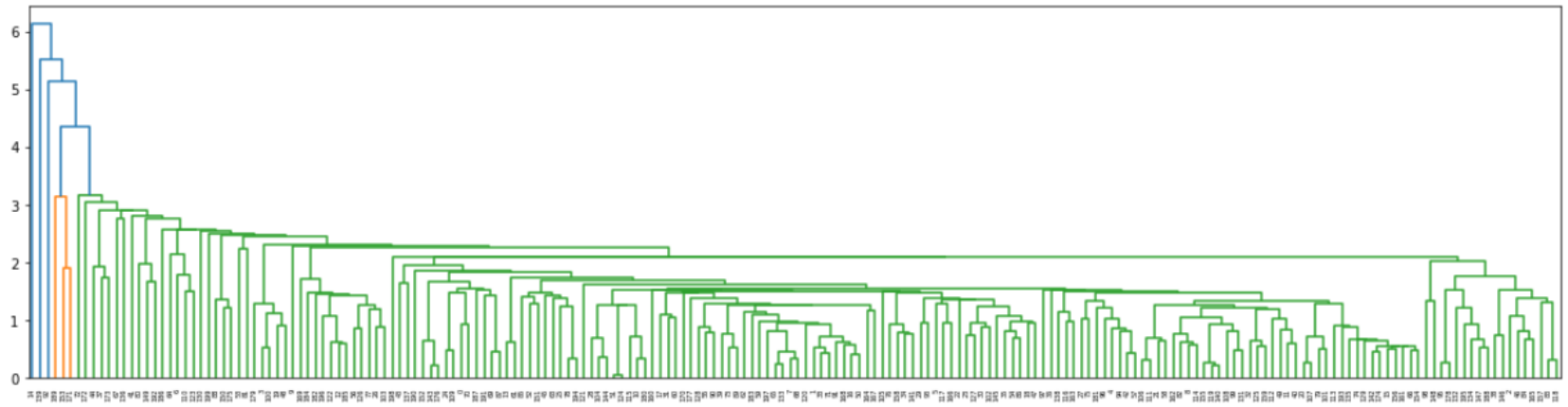
```
In [8]: agglo = AgglomerativeClustering(n_clusters=2)
        agglo.fit(X2)
        myColors = {0:'red',1:'green'}                                    # Define a color palette: 0~1.
        plt.scatter(X2[:,0],X2[:,1], c= pd.Series(agglo.labels_).apply(lambda x: myColors[x]), alpha=0.7 )
        plt.title('Dataset #2 : Agglomerative')
        plt.show()
```



Apply agglomerative clustering and visualize

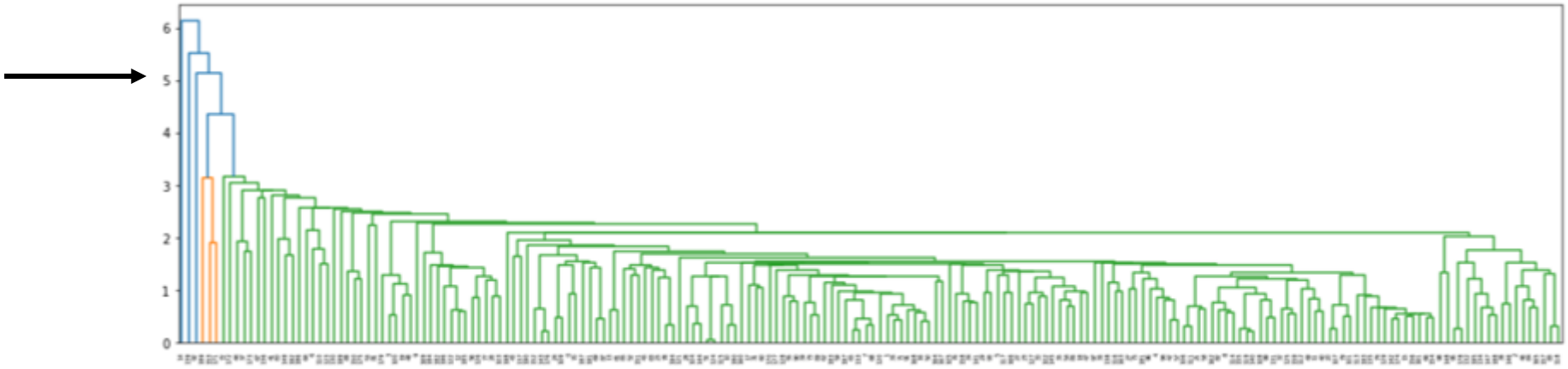
- Dataset #1 and show dendrogram.
- Cluster hierarchically using single linkage

```
In [9]: from scipy.cluster.hierarchy import linkage, dendrogram, fcluster

In [10]: myLinkage = linkage(X1,method='single')
         plt.figure(figsize=(20,5))
         dendrogram(myLinkage)
         plt.show()
```



Apply agglomerative clustering and visualize

- Dataset #1 and show dendrogram.
- Cut at the height (distance) = 5 (change this value at will).

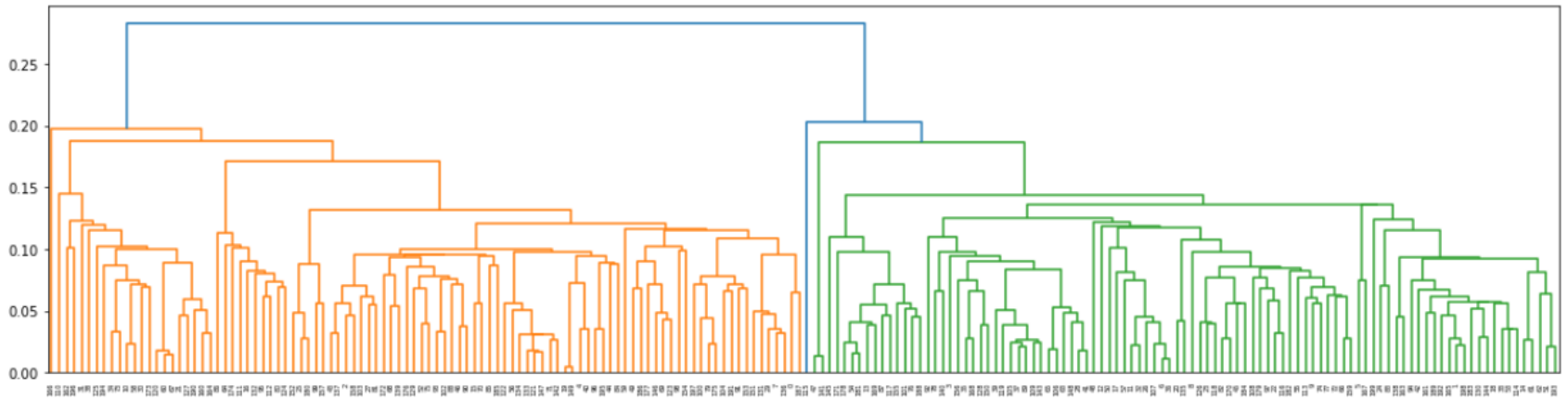



```
In [11]: labels = fcluster(myLinkage, 5, criterion='distance')
         pd.Series(labels).value_counts()

Out[11]: 1    197
         2      1
         3      1
         4      1
         dtype: int64
```

Apply agglomerative clustering and visualize

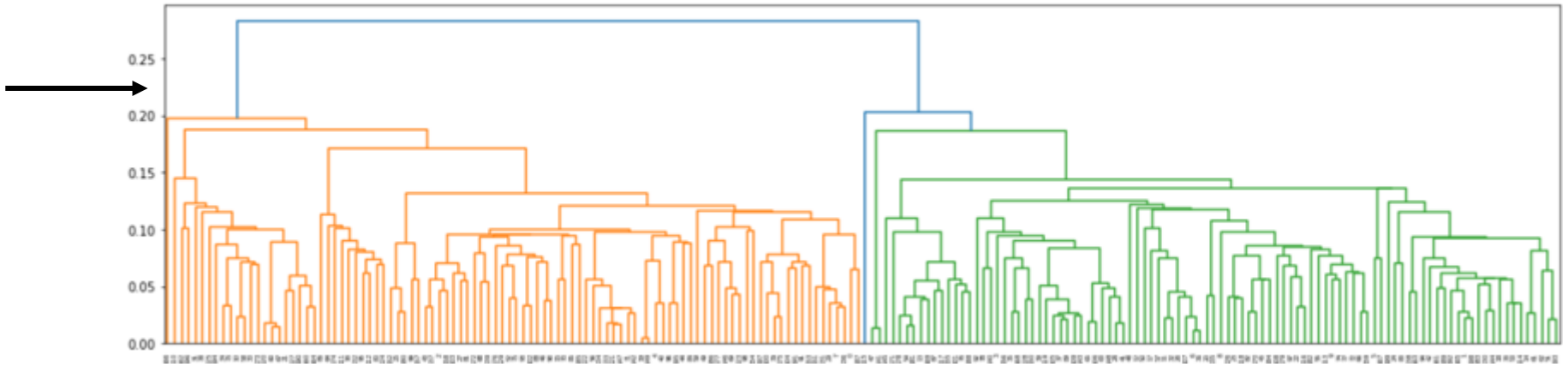
- Dataset #2 and dendrogram
- Cluster hierarchically using single linkage

```
In [12]: myLinkage = linkage(X2,method='single')
         plt.figure(figsize=(20,5))
         dendrogram(myLinkage)
         plt.show()
```



Apply agglomerative clustering and visualize

- Dataset #2 and clusters by cutting the dendrogram
- Cut at the height (distance) = 0.23 (change this value at will)

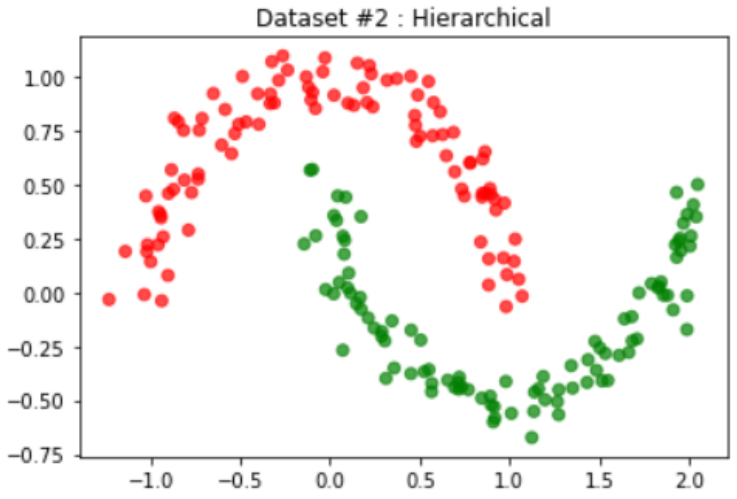



```
In [13]: labels = fcluster(myLinkage, 0.23, criterion='distance')
         pd.Series(labels).value_counts()

Out[13]: 1    100
         2    100
         dtype: int64
```

Apply agglomerative clustering and visualize

- Dataset #2: define a color palette: 1~2

```
In [14]: myColors = {1:'red',2:'green'}
         plt.scatter(X2[:,0],X2[:,1], c= pd.Series(labels).apply(lambda x: myColors[x]), alpha=0.7 )
         plt.title('Dataset #2 : Hierarchical')
         plt.show()
```



Apply agglomerative clustering and visualize

- Iris dataset

In [4]:
```
iris_data=iris.data
iris_data_pd=pd.DataFrame(iris.data, columns=iris.feature_names)
iris_data_pd
```

Out[4]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

Apply agglomerative clustering and visualize

- Dataset #2: working with petal features

```
In [5]: iris_data_pd.iloc[:,2:4]
```

Out[5]:

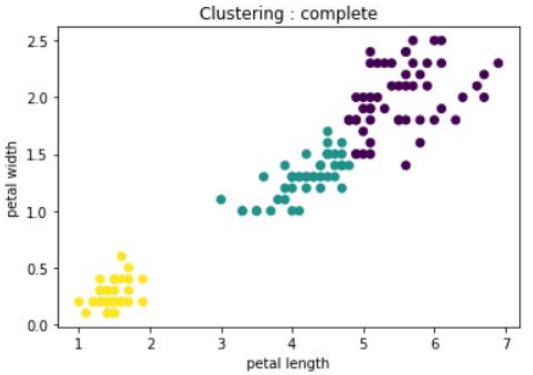
	petal length (cm)	petal width (cm)
0	1.4	0.2
1	1.4	0.2
2	1.3	0.2
3	1.5	0.2
4	1.4	0.2
...	...	...
145	5.2	2.3
146	5.0	1.9
147	5.2	2.0
148	5.4	2.3
149	5.1	1.8

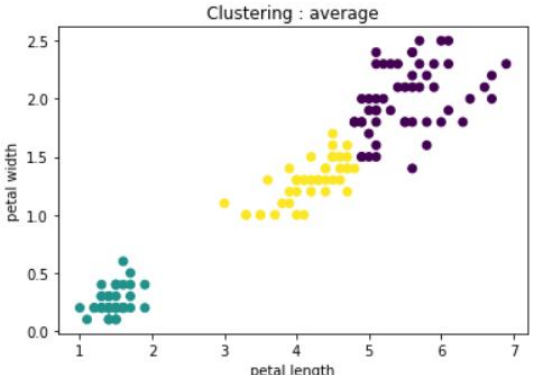
150 rows × 2 columns

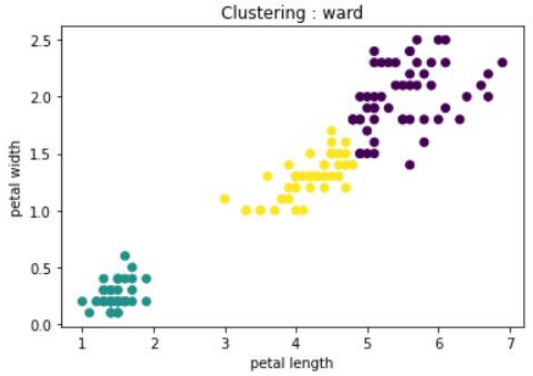
Apply agglomerative clustering and visualize

- To use the 'petal length' and 'petal width' columns, import the sklearn agglomerative clustering, which is supported by the sklearn.clustering package.
- The distance measurement criteria between clustering can be set with the linkage parameter, which only supports ward, complete, and average. Check out how all three cases are used.

```
In [6]: from sklearn.cluster import AgglomerativeClustering
        likage=["complete", "average","ward"]
        for idx, i in enumerate(likage):
            plt.figure(idx)
            hier=AgglomerativeClustering(n_clusters=3, affinity='euclidean', linkage=i)
            hier.fit(iris_data_pd.iloc[:,2:4])
            plt.scatter(iris_data_pd.iloc[:,2], iris_data_pd.iloc[:,3],c=hier.labels_)
            plt.title("Clustering : " + i)
            plt.xlabel('petal length')
            plt.ylabel('petal width')
            plt.show()
```



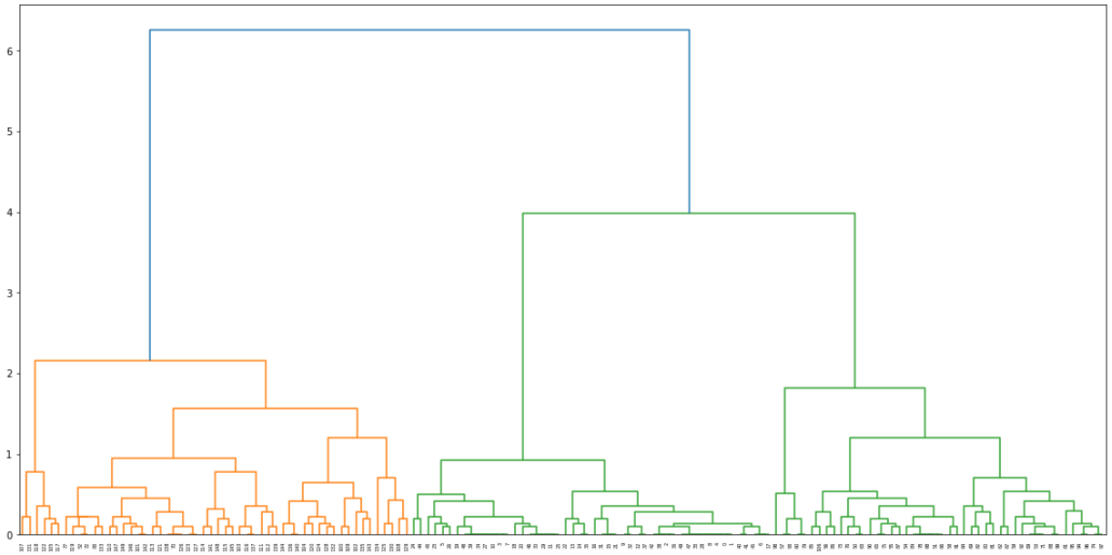




Apply agglomerative clustering and visualize

- Use Scikit-learn to confirm the clustering result. You may want to check how the tree is created, but this package does not provide this function.
- Instead, use the Scipy package for clustering and drawing a tree.
- As provided previously, implement the **complete** linkage

```
In [7]: from scipy.cluster import hierarchy

In [8]: hierar=hierarchy.linkage(iris_data_pd.iloc[:,2:4],'complete')
        plt.figure(figsize=(20,10))
        dn=hierarchy.dendrogram(hierar)
```



Apply agglomerative clustering and visualize

- Use Scikit-learn to confirm the clustering result. You may want to check how the tree is created, but this package does not provide this function.
- Instead, use the Scipy package for clustering and drawing a tree.
- As provided previously, implement the **single** linkage

```
In [9]: hierar=hierarchy.linkage(iris_data_pd.iloc[:,2:4],'single')
        plt.figure(figsize=(20,10))
        dn=hierarchy.dendrogram(hierar)
```

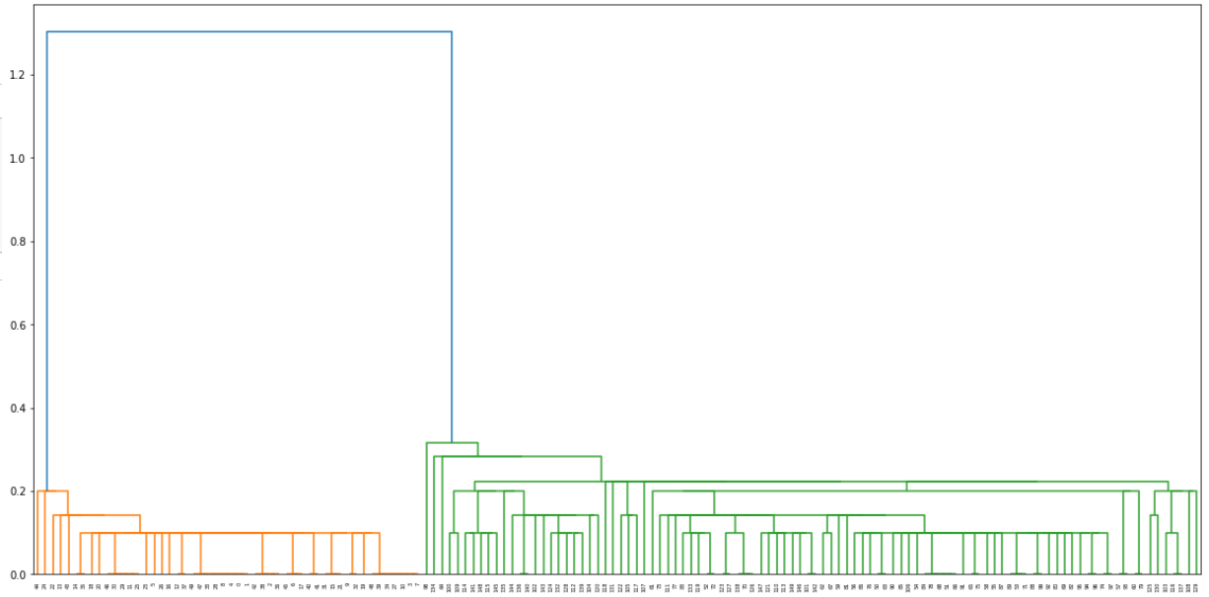
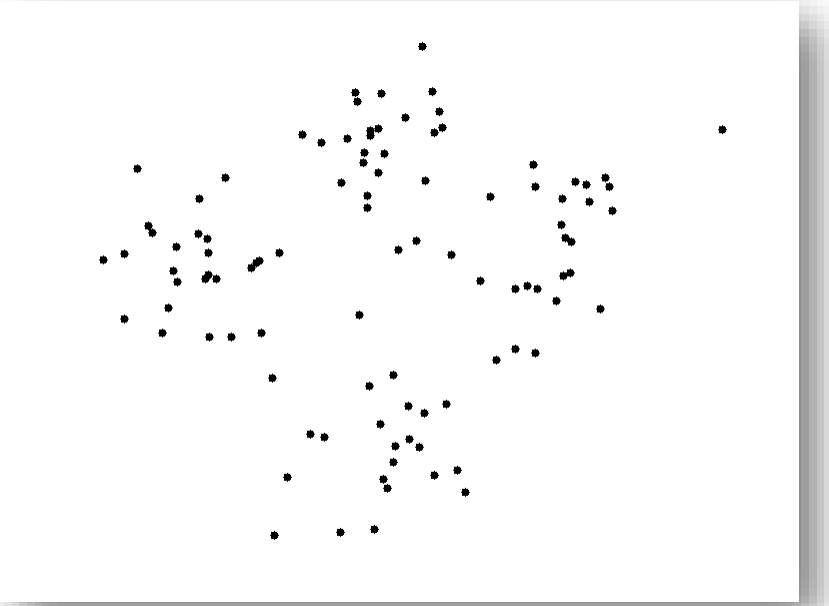


Table of contents

Partitional clustering

- Partitioning method: Construct a partition of a database D of n objects into a set of k clusters
  - Given a value for k, find a partition of k clusters that optimizes the chosen partitioning criterion
  - Global optimal: exhaustively enumerate all possible partitions
  - Heuristic methods: k-means and k-medoids algorithms
    - k-means (MacQueen'67): each cluster is represented by the center of the cluster
    - k-medoids or PAM (Partition around medoids) (Kaufman & Rousseeuw'87): each cluster is represented by one of the objects in the cluster

1: Select K points as the initial centroids.
2: **repeat**
3: Form K clusters by assigning all points to the closest centroid.
4: Recompute the centroid of each cluster.
5: **until** The centroids don't change



A deeper insight on K-Means

- Initial centroids are often chosen randomly
  - Clusters produced vary from one run to another
  - The centroid is (typically) the mean of the points in the cluster

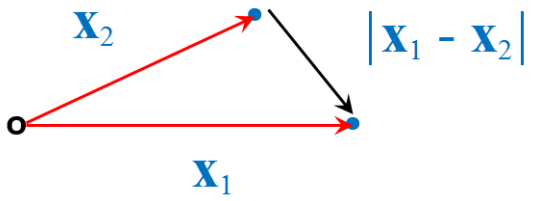
- K-means will converge for common similarity measures
  - Most of the convergence happens in the first few iterations
  - Often the stopping condition is changed to 'Until relatively few points change clusters'
  - Often terminates at a *local optimum*. Heuristic techniques should be used to find the global optimum

- Complexity is $O(n \cdot K \cdot I \cdot d)$
  - n = number of points, K = number of clusters, I = number of iterations, d = number of attributes

- Weaknesses
  - Applicable only when mean is defined, then what about categorical data?
  - Need to specify k, the number of clusters, in advance
  - Unable to handle noisy data and outliers
  - Not suitable to discover clusters with non-convex shapes

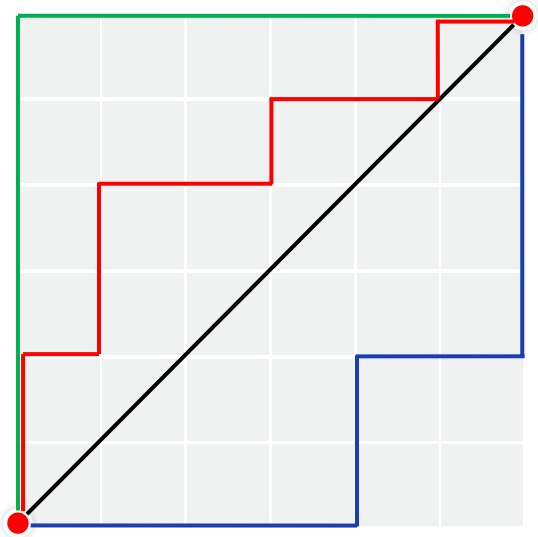
Distance metric

- Euclidean distance: it is the modulus of the difference between two position vectors



$$|X_1 - X_2| = \sqrt{(x_{11} - x_{21})^2 + (x_{12} - x_{22})^2 + \cdots + (x_{1d} - x_{2d})^2}$$

- For numeric variables:
  - Euclidean, Standardized, Mahalanobis, Chebyshev, Canberra, Manhattan, Minkowski, etc.
- For categorical variables:
  - Jackard, etc.



Black: Euclidean / Colored: Manhattan

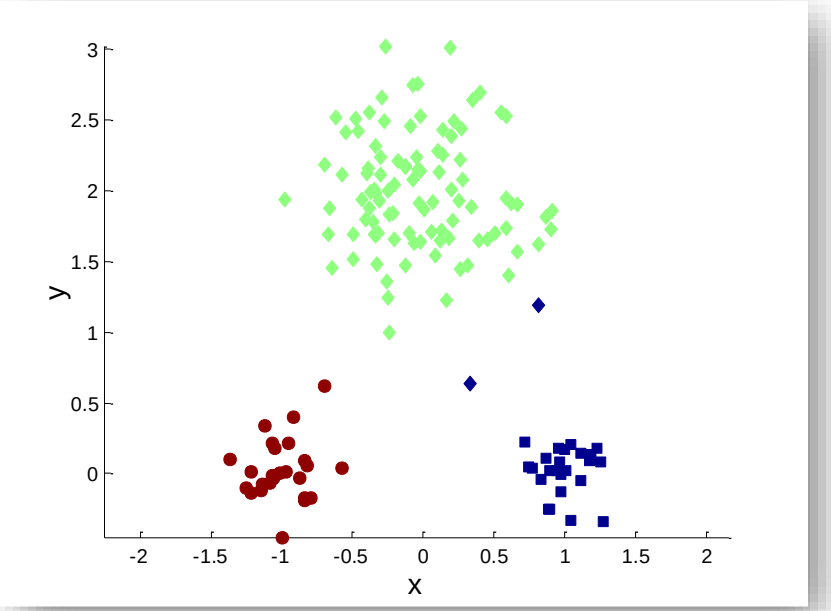
Santard algorithm

1) Let's suppose a dataset composed of n observations: $\boldsymbol{x}1, \boldsymbol{x}2, \cdots, \boldsymbol{x}n$
   Each observation $\boldsymbol{x}i$ is a vector of d components (d = number of variables)

2) Suppose that we target two (k=3) clusters, denoted as $C1$, C2 and $C3$

3) Two centroid positions are randomly initialized: $\mu 1$, $\mu 2$ and $\mu 3$

4) Assign the observations to the nearest centroids



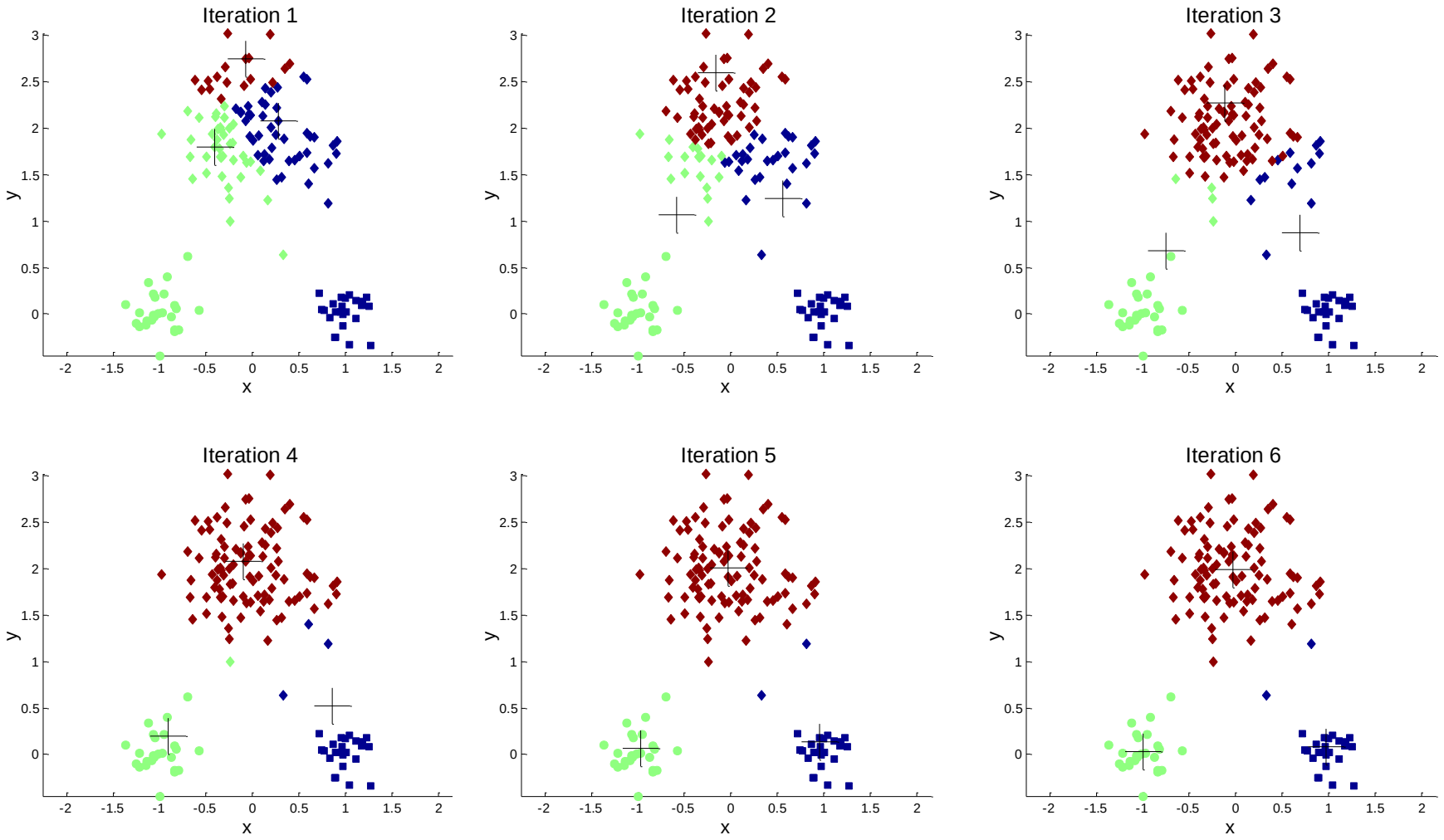
5) Update the centroid positions, such that the sum of square distances between the observations and the nearest centroids gets smaller

$$minimize \sum_{i=1}^{k} \sum_{x \in C_i} |\boldsymbol{x} - \boldsymbol{\mu}_i|^2$$

6) Repeat from step 4) until the centroids converge to their final positions

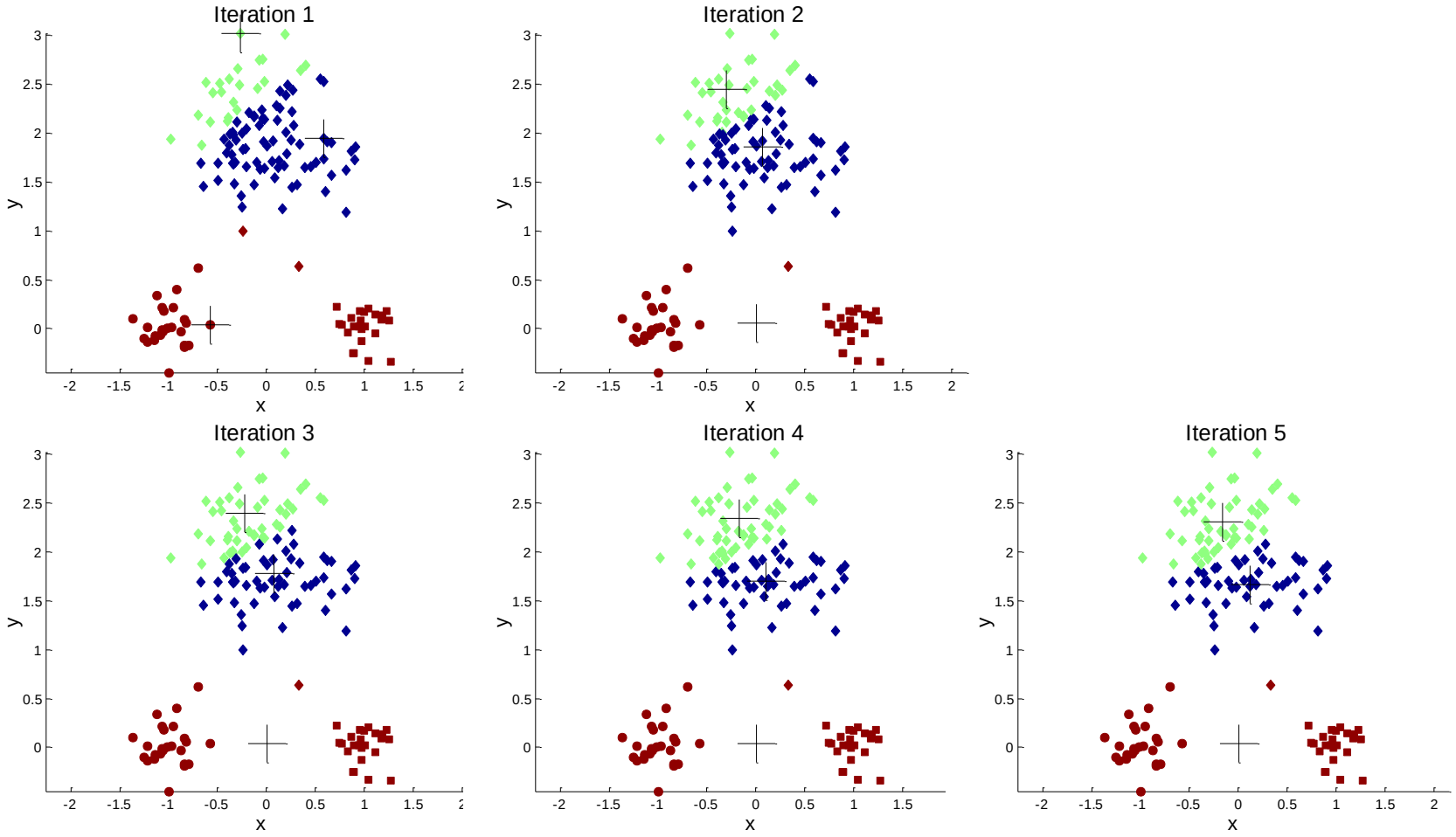
Santard algorithm

K-Means: Inicialization #1



Santard algorithm

K-Means: Inicialization #2



Workarounds:

1) Multiple runs

2) Sample and use hierarchical clustering to determine initial centroids

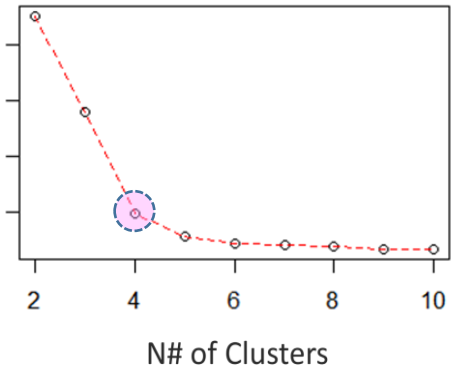
3) Select more than k initial centroids and then select among these initial centroids those k most widely separated

4) Apply a meta-heuristic search

A question: how do we find the optimal number of clusters?

- Elbow method
  - Let's define the sum of square distances between the observations and the nearest centroids as:

$$Total\ internal\ sum\ of\ squares = \sum_{i=1}^{k} \sum_{x \in C_i} |\boldsymbol{x} - \boldsymbol{\mu}_i|^2$$



  - Plot the "Total internal sum of squares" vs "N# of clusters" and find the inflexion point.

- Silhouette coefficient
  - The silhouette of a datum is a measure of how closely it is matched to data within its cluster and how loosely it is matched to data of the neighboring cluster
  - A silhouette close to 1 implies the datum is in an appropriate cluster, while a silhouette close to -1 implies the datum is in the wrong cluster
  - Optimization techniques such as evolutionary algorithms are useful in determining the number of clusters that gives rise to the largest silhouette

- Others: e.g. affinity propagation, mean shift (replaces the problem by another)

K-Means: problems with non-globular datasets

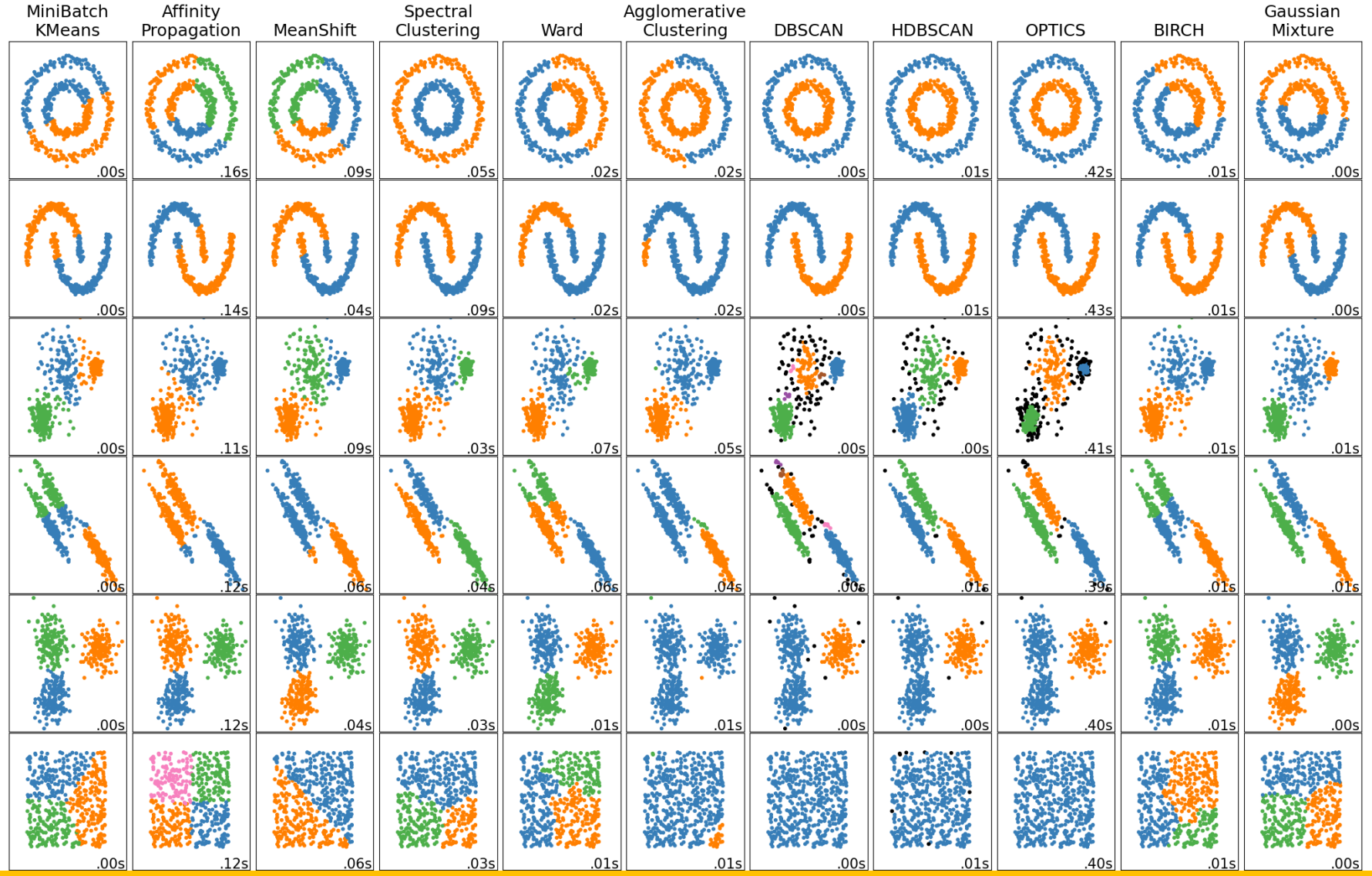


Table of contents

Practical Notes

- Clustering-1.ipynb
  - K-Means clustering with simulated data.
    i. Generate simulated data and visualize.
    ii. Apply k-means clustering and visualize.

- Clustering-2.ipynb
  - *K-Means clustering with real data*
    i. Load the 'Iris' dataset from Seaborn
    ii. Apply k-means
    iii. Visualize
    iv. Prediction based on what we have learned

Table of contents

Other clustering methods

- K-medoids clustering
  - K-medoids clustering is the advanced version of the K-means clustering method. It uses all types of similarity and dissimilarity measures. Moreover, it is advantageous for noise or outlier processing because it designates the cluster center using actual data set values rather than random dots on plane coordinates.

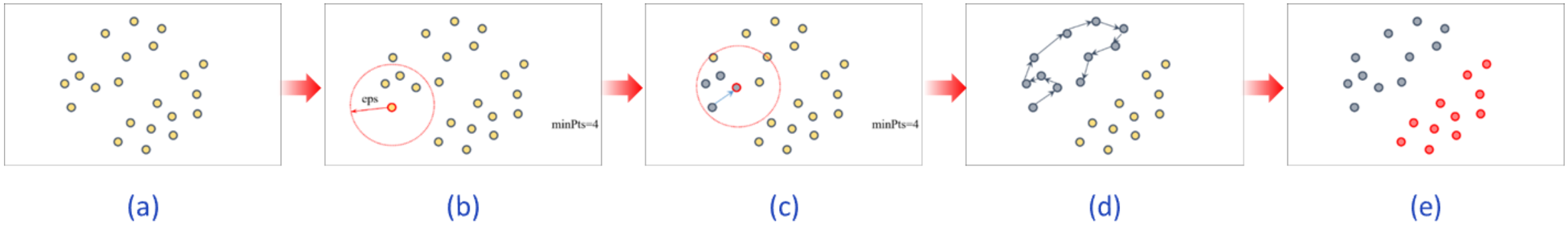
- DBSCAN (Density Based Spatial Clustering of Application with Noise)
  - K-means clustering method calculates the average of K-clusters and distances between each data point for clustering. On the other hand, DBSCAN applies density to create the same group of data sets linked with constant density. It is a clustering method that is advantageous for identifying noise and outliers.

- Gaussian Mixture Model
  - It is a probability-based clustering analysis that predicts the parameter using the EM (Expectation Maximization) or MCMC (Markov Chain Monte Carlo) algorithms.

Density-based spatial clustering of applications with noise (DBSCAN)

- The most common algorithm in density clustering is the Density-based Spatial Clustering of Applications with Noise (DBSCAN) algorithm. Similar to K-means clustering, DBSCAN clustering is affected by data distribution.
- While K-means clustering creates clusters according to the distribution using the standard data, the DBSCAN clustering is affected by the density of each data. In other words, it assumes that the data included in the same cluster would have high density.
- Pros
  - It can reveal structures that the k-means and hierarchical clustering cannot.
- Cons
  - Hyperparameters eps and minPts must be specified.
  - It is hard to get stable clusters in case the dense regions overlap

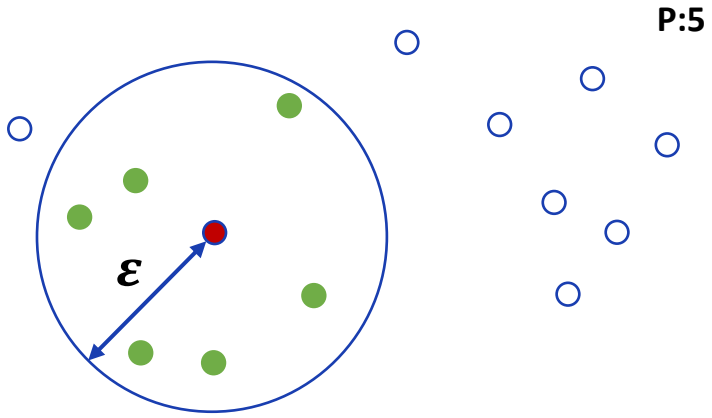
DBSCAN algorithm



1. Suppose that the observations are distributed as in (a).
2. From a point, count the points within a radius eps and look around: (b).
3. If there are more than minPts points within the circle, include them in the same cluster: (b).
4. Then move on to the next point and repeat from step 2) until all the points are exhausted: (c)~(d).
5. Move on to another point that has not been clustered yet. Repeat from step 2): (e).

How to determine density for DBSCAN algorithm

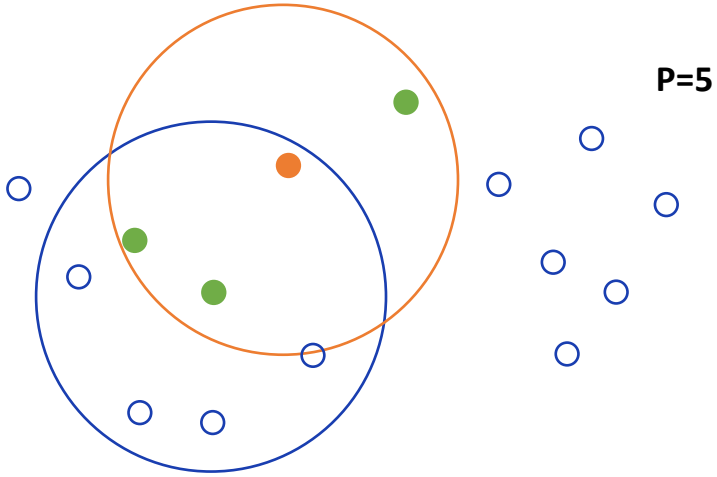
- Check the following parameters to determine the density for DBSCAN algorithm.
  1. ε as the radius of the circle with the element in the center
  2. P as the minimum number of elements that exist within the radius. The radius distance is measured with the Euclidian distance method.
- Look at the following figure. First, select a random point.



How to determine density for DBSCAN algorithm

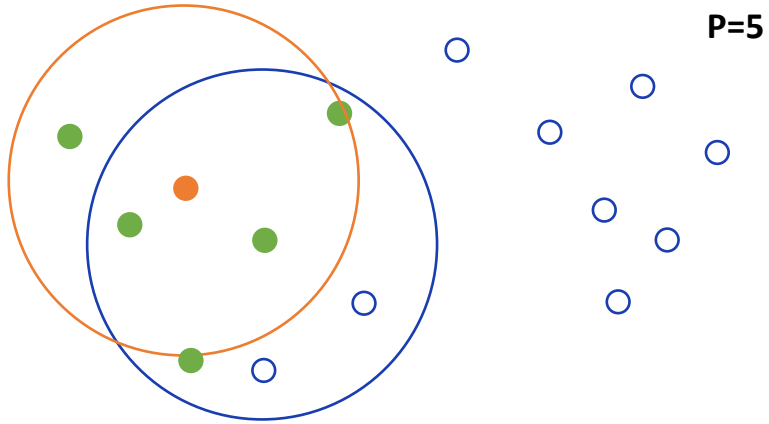
- The randomly selected point is designated at a red dot, as shown in the figure in the previous slide.
- The circle is centered on the red dot and has a radius of ε. Check if the number of elements inside the circle is the same as P or greater than P.
- If it is the same or greater than P, then the red dot becomes a core object, and the circle is created as a cluster. If the value is smaller than P, the red dot is defined as noise.
- In the figure, the elements in the red circle are six green dots. P equals 5, so the minimum number of elements that should exist within the radius is satisfied.
- Thus, the red dot becomes a core object, and the red circle becomes a cluster. Now, move on to other elements. Select one of the elements in the circle other than the core object.
- Let's look at the new dot in the next figure.

How to determine density for DBSCAN algorithm



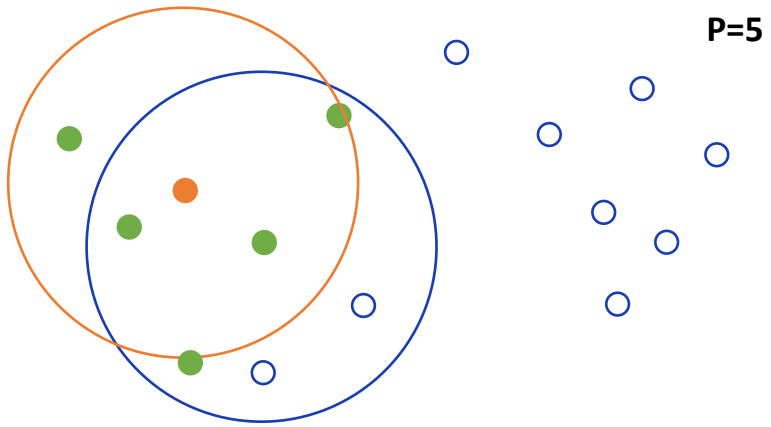
- Blue circle is the existing cluster, and a new dot is designated within the radius. When creating a circle with the red dot in the center, only three elements are inside it.
- The P value is 5, so the number of elements inside the circle is less than P. Thus, this dot cannot become the core object. Instead, it is defined as noise.
- Move on to another dot within the radius.

How to determine density for DBSCAN algorithm



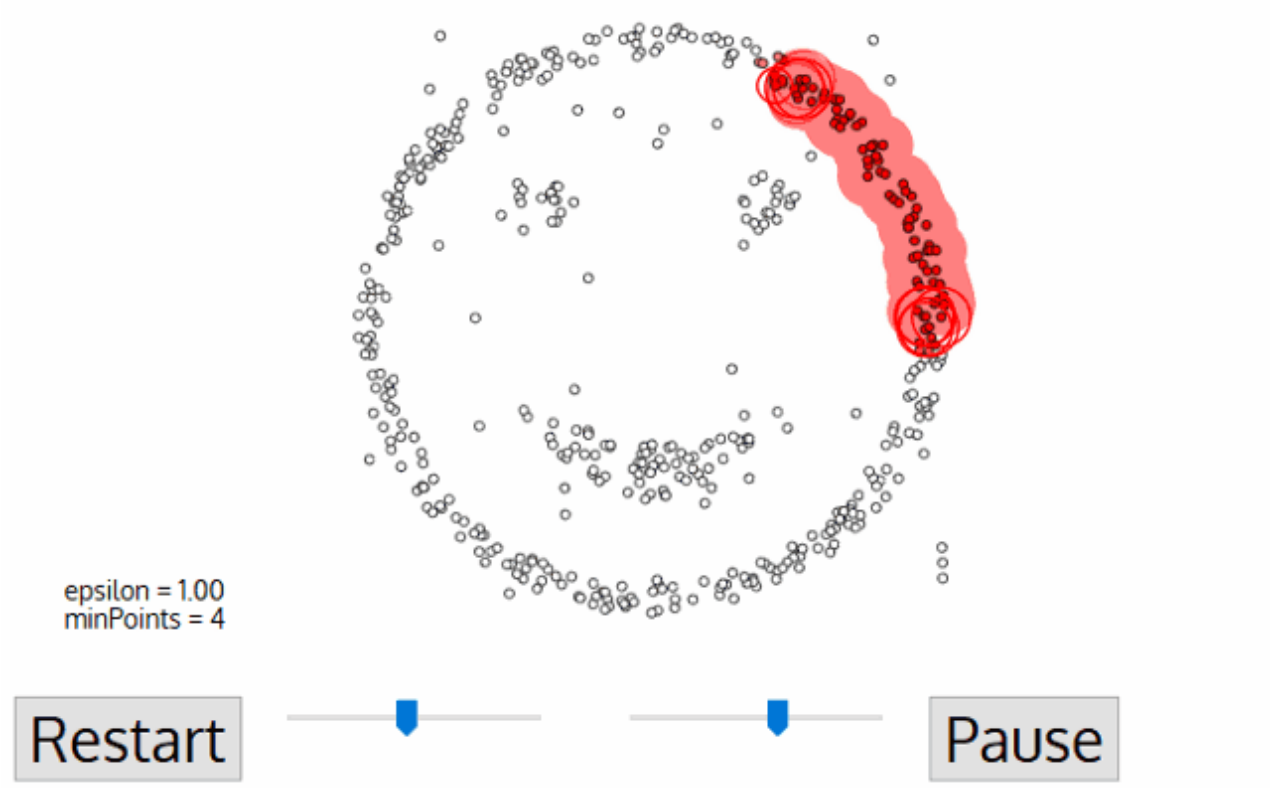
- Now, there are 5 elements in the red circle. It is the same as the P value, so this red dot is recognized as a core object.
- It is possible to create a cluster, but make sure to consider if this element was included in the previous cluster, as it is now classified as a new core object.
- If included in the previous cluster, do not create a new cluster but expand the existing cluster. If it is not relevant to the previous cluster, a new cluster will be created using the red dot as a new core object.
- In the figure, the red dot is an element presented in the blue circle, which is the previous cluster. Since it is included in the previously created cluster, the previous cluster is expanded without making a new cluster.

How to determine density for DBSCAN algorithm



- This is the expanded cluster.
- The DBSCAN algorithm aims to check every dot by repeating the same process over and over. After completing all processes, the data cluster looks as follows.

How to determine density for DBSCAN algorithm



Practical Notes

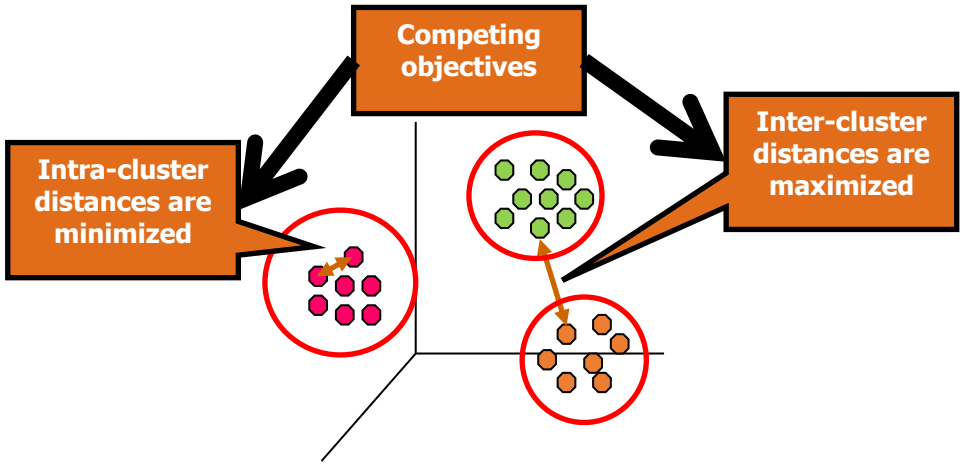
- Clustering-3.ipynb
    - Compare clustering algorithms.
        i. Generate simulated data and visualize.
        ii. Apply k-means clustering and visualize.
        iii. Apply agglomerative clustering and visualize
        iv. Apply hierarchical clustering and visualize.
        v. Apply DBSCAN and visualize

Evaluation of a clustering structure

- Based on internal structural information

1.Intra-cluster cohesion (compactness):
- Cohesion measures how near the data points in a cluster are to the cluster centroid
- Sum of squared error (SSE) is a commonly used measure

2.Inter-cluster separation (isolation):
- Separation means that different cluster centroids should be far away from one another



✓ In most applications, expert judgments are still the key

✓ In some applications, clustering is **not the primary task**, but used to help perform another task (e.g. book recommendation systems)

✓ We can use the performance on the primary task to compare clustering methods
  ✓ Often subject to a complex clustering performance inference

Evaluation of a clustering structure

Examples for partitional clustering

- **Calinski-Harabasz:** ratio between intra-cluster and inter-cluster dispersion:
  - The higher the value is, the better the discovered set of clusters can be claimed to be

$$\mathrm{CH}(P) = \frac{(N - |P|)\,\mathrm{inter}_{CH}(P)}{(|P| - 1)\,\mathrm{intra}_{CH}(P)}$$

$$\mathrm{inter}_{CH}(P) = \sum_{C \in P} |C|\,\mathrm{d}(\bar{C}, \bar{X}) \text{ e } \mathrm{intra}_{CH}(P) = \sum_{C \in P} \sum_{x \in C} \mathrm{d}(x, \bar{C})$$

N is the number of instances
$|P|=k$ is the number of clusters

- **Silhouette:**
  - Let a (i) be the mean distance of object i from the rest of the objects in its cluster. Let b(i) be the minimum mean distance of object i to the rest of the objects of the rest of clusters
  - s (i) takes values in [-1,1], being better the closer it is to 1. If s (i) approaches zero, it means that the object is on the edge of two clusters
  - Then we compute, for every object:

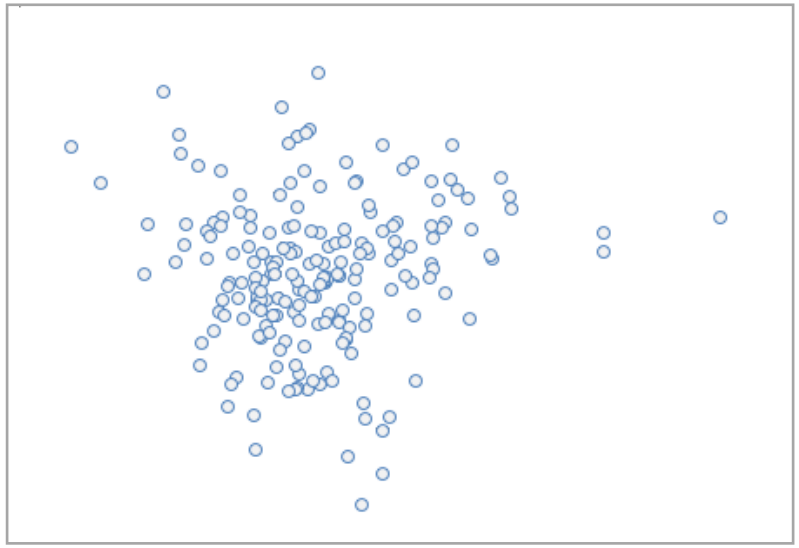
$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

  - The mean of all s (i) is the silhoutte coefficient that measures the overall quality of the clustering

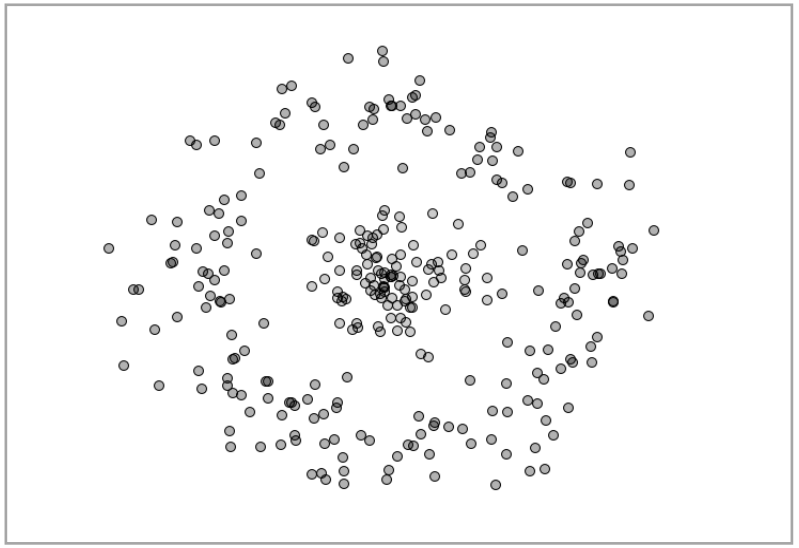
How to choose a clustering algorithm

- Choosing the “best” algorithm is a challenge!!!
  - Every algorithm has limitations and works well with certain data distributions
  - It is very hard, if not impossible, to know what distribution the application data follow.
    - The data may not fully follow any “ideal” structure or distribution required by the algorithms.
- Due to these complexities, the common practice is to run several algorithms using different distance functions and parameter settings, and then carefully analyze and compare the results

Which clustering algorithms are suitable for the following cases?



(a)



(b)

Final recommendations

- Clustering has along history and still very active (e.g. clustering ensembles)
- Clustering is very useful in practice, yet hard to evaluate
- Clustering is highly application dependent and to some extent, subjective
- My personal recommendation for clustering (and data mining) newcomers:

A comparison of the clustering algorithms in scikit-learn

Method name	Parameters	Scalability	Usecase	Geometry (metric used)
K-Means	number of clusters	Very large `n_samples`, medium `n_clusters` with *MiniBatch code*	General-purpose, even cluster size, flat geometry, not too many clusters	Distances between points
Affinity propagation	damping, sample preference	Not scalable with n_samples	Many clusters, uneven cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Mean-shift	bandwidth	Not scalable with `n_samples`	Many clusters, uneven cluster size, non-flat geometry	Distances between points
Spectral clustering	number of clusters	Medium `n_samples`, small `n_clusters`	Few clusters, even cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Ward hierarchical clustering	number of clusters	Large `n_samples` and `n_clusters`	Many clusters, possibly connectivity constraints	Distances between points
Agglomerative clustering	number of clusters, linkage type, distance	Large `n_samples` and `n_clusters`	Many clusters, possibly connectivity constraints, non Euclidean distances	Any pairwise distance
DBSCAN	neighborhood size	Very large `n_samples`, medium `n_clusters`	Non-flat geometry, uneven cluster sizes	Distances between nearest points
Gaussian mixtures	many	Not scalable	Flat geometry, good for density estimation	Mahalanobis distances to centers