

Characteristic Patterns in Data and (Unsupervised) Classification: Cluster Analysis.

Cluster Analysis:

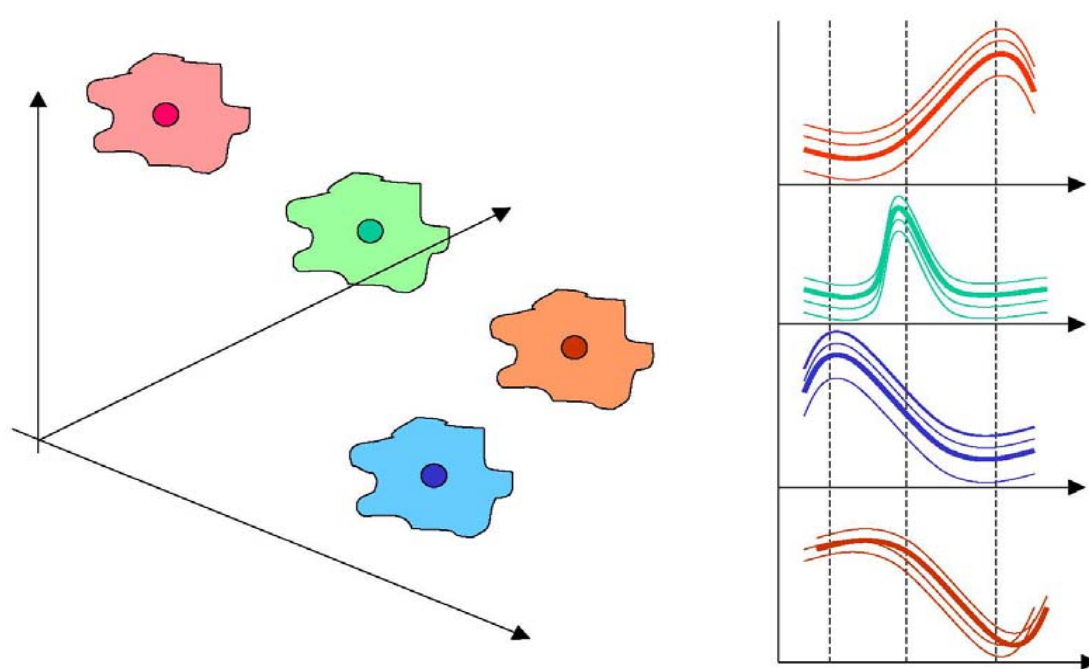
Identify groups of observations that are similar with respect to a certain number of variables/features. Similarity is defined through a distance, in a space where observations are points, and dimensions are variables recorded on the observations.

Given a preprocessed microarray data matrix, it may be useful to cluster:

- Columns, when they correspond to **samples**. n (or T , # of columns) observations and p (or N , # of rows) variables, i.e. n points in an p -dimensional space. Unsupervised classification (“class discovery”) for the samples, based on gene expression. Typical in cancer studies.
- Rows, which correspond to **genes**. N observations and T variables, i.e. N points in an T -dimensional space. **Partition genes** into classes presenting similar expression profiles over the conditions of interest, and **extract characteristic expression patterns** as cluster centroids.

Rationale: genes with similar expression profiles may be involved in similar/related functions, and possibly be co-regulated. WARNING: expression data by itself may not be enough to reliably investigate co-regulation.

(sometimes cluster both genes *and* samples; plaid schemes, Lazzeroni and Owen 02)



Basic cartoon: partition and characteristic patterns

Pre-processing of the data matters, for instance:

- Center and/or standardize by row? (then a typical Euclidean distance will score as similar profiles with similar shapes, regardless of their location and scale).
- Center and/or standardize by column? (then a typical Euclidean distance will be insensitive to original differences in variability scales for different “conditions”)

Literature on clustering is huge (data mining). We consider only algorithms that:

- create hard partitions (**Exclusive**; overlapping clusters are not allowed; algorithms allowing for overlapping or soft partitions do exist).
- do not use information on the possible groups (**Unsupervised**; intrinsic)

Hierarchical Clustering (agglomerative)

Set of N data points in \mathbf{R}^T .

- Choose a distance function for points $d(x_1, x_2)$ (Euclidean distance; correlation distance; other more complicated distances). Sometimes the point distance is not defined explicitly as a function, but provided through an $N \times N$ matrix.
- Choose a distance function for clusters $D(C_1, C_2)$ based on a summary of the distances between points, as measured by $d(x_1, x_2)$ (for clusters formed by just one points, D reduces to d).

Proceeding in an agglomerative fashion (“bottom-up”), generate a sequence of nested partitions of the data – progressively less fine:

- Start from N clusters, each containing one data point.
- At each iteration:
 1. Using the current matrix of cluster distances, find the two closest clusters.
 2. Update the list of clusters by merging the two closest.
 3. Update the matrix of cluster distances accordingly
- Repeat until all data points are joined in one cluster.

Remarks:

- The method is sensitive to anomalous data points/outliers
- Mergers are irreversible: “bad” mergers occurring early on affect the structure of the nested sequence (path dependence).
- If two pairs of clusters are equally (and maximally) close at a given iteration, we have to choose arbitrarily; the choice will affect the structure of the nested sequence.

Defining cluster distance: the linkage function

$$D(C_1, C_2) = \min_{x_1 \in C_1, x_2 \in C_2} d(x_1, x_2)$$

Single (string-like, long)

$$D(C_1, C_2) = \frac{1}{\#(C_1)\#(C_2)} \sum_{x_1 \in C_1, x_2 \in C_2} d(x_1, x_2)$$

Average

$$D(C_1, C_2) = \max_{x_1 \in C_1, x_2 \in C_2} d(x_1, x_2)$$

Complete (ball-like, compact)

$$D(C_1, C_2) = d(\bar{x}_1, \bar{x}_2)$$

Centroid

Single and complete linkages produce nested sequences invariant under monotone transformations of d – not the case for average linkage. However, the latter is a compromise between “long”, “stringy” clusters produced by single, and “round”, “compact” clusters produced by complete.

Example

Agglomeration step in constructing the nested sequence (first iteration):

Left: matrix of distances among 5 data points. 3 and 5 are the closest, and are therefore merged in cluster “35”.

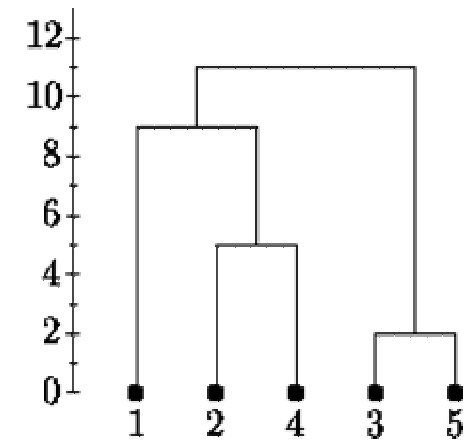
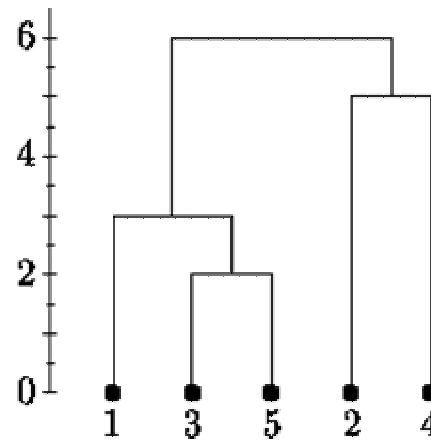
Right: new distance matrix computed with complete linkage.

	1	2	3	4	5
1	0				
2	9	0			
3	3	7	0		
4	6	5	9	0	
5	11	10	2	8	0

	35	1	2	4
35	0			
1	11	0		
2	10	9	0	
4	9	6	5	0

Dendrogram representing the nested sequence produced by single linkage (left) and complete linkage (right).

Ordinate: distance, or height, at which each merger occurred. Horizontal ordering of the data points is any order preventing intersections of branches.



IMPORTANT: hierarchical clustering, per se, does not dictate a partition and a number of clusters; it provides a nested sequence of partitions (this is more informative than just one partition). To settle on one partition, we have to “**cut**” the dendrogram. Loosely: read the height associated with various numbers of clusters as a measure of “fit” of the partitioning in K groups (more later).

Partitioning algorithms: K-means.

Set of N data points in \mathbf{R}^T .

- Choose a distance function for points $d(x_1, x_2)$.
- Choose K = number of clusters.
- Initialize the K cluster centroids (e.g. K data points chosen at random) $\bar{x}_1(0), \dots, \bar{x}_K(0)$

Use the data to iteratively relocate centroids, and reallocate points to closest centroid.

- At each iteration:

1. Compute distance of each data point from each current centroid.

$$d(x, \bar{x}_k) \quad \forall x, k = 1 \dots K$$

2. Update current cluster membership of each data point, selecting the centroid to which the point is closest.

$$m(x) = \operatorname{argmin}_{k=1 \dots K} d(x, \bar{x}_k) \quad \forall x$$

3. Update current centroids, as averages of the new clusters formed in 2.

$$\bar{x}_k = \frac{1}{\#(x : m(x) = k)} \sum_{x:m(x)=k} x \quad k = 1 \dots K$$

- Repeat until cluster memberships, and thus centroids, stop changing.

Remarks:

- Also this method is sensitive to anomalous data points/outliers.
- Points can move from one cluster to another, but the final solution depends strongly on centroid initialization (another form of path dependence) .
- If two centroids are equally (and maximally) close to an observation at a given iteration, we have to choose arbitrarily (the problem here is not so serious because points can move...).
- There are several “variants” of the k-means algorithm.

Objective function

K-means converges to a local minimum of the **total within-cluster square distance** (total within cluster sum of squares) – not necessarily a global one.

$$\Delta(K, start) = \sum_{k=1}^K \sum_{x:m(x)=k} d^2(x, \bar{x}_k)$$

$$= \sum_x \min_{k=1 \dots K} d^2(x, \bar{x}_k)$$

$$= \sum_x \sum_{k=1}^K \text{Ind}(m(x) = k) d^2(x, \bar{x}_k)$$

Second form: centroids as “optimization vrb'l's” (memberships can be derived from them; see how final value of the obj fct depends on centroid initialization).

Third form: useful for comparison with SOM and fuzzy k-means.

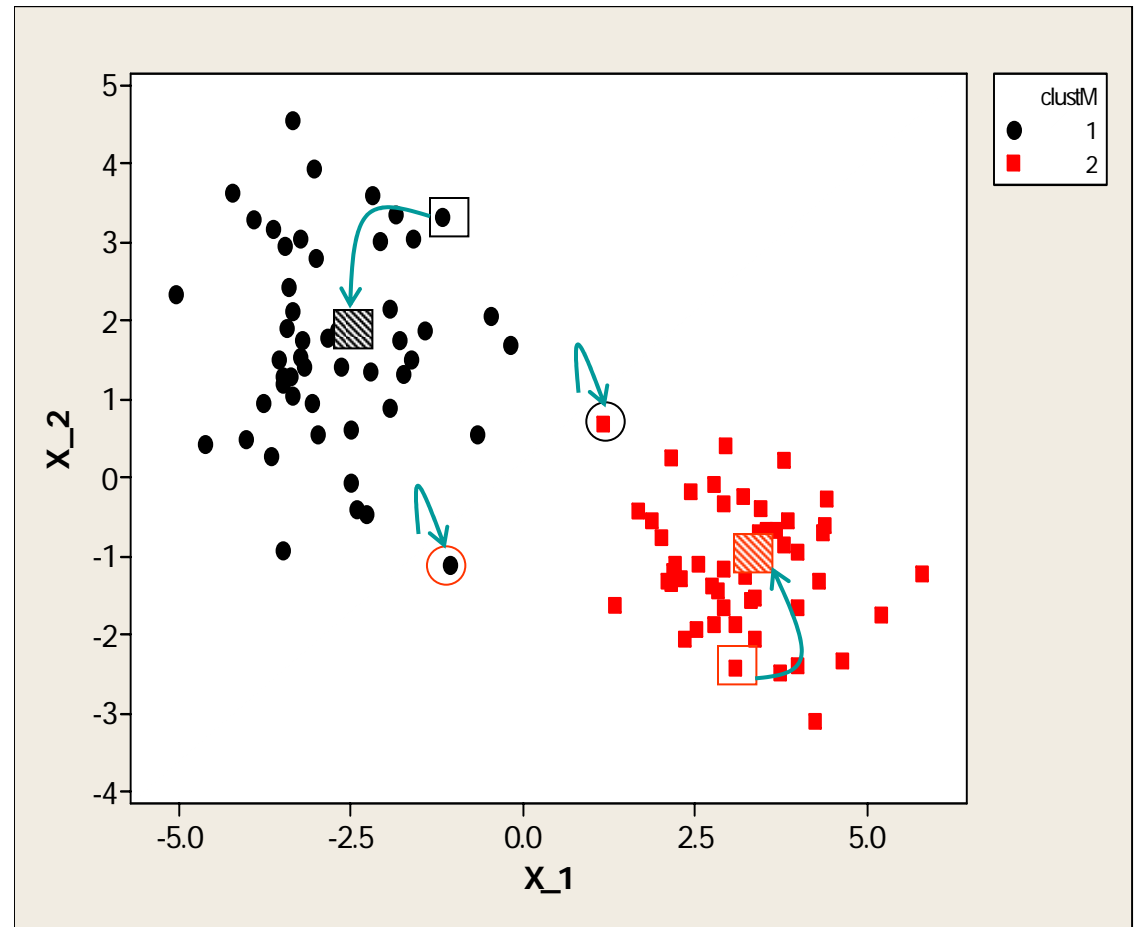
Clusters tend to be ball-shaped with respect to the chosen distance.

Example

Iteration step (first) in updating centroids and memberships.

Rectangles: random centroid initializations, and updated centroids.

Circles: updated memberships.



IMPORTANT: K-means creates only one partition with the specified number of clusters; to get partitions with different K 's need to run the algorithm again (and nesting is not guaranteed). Loosely: read the final sum of squares value associated with various numbers of clusters as a measure of “fit” of the partitioning in K groups (more later).

Variants on partitioning algorithms:

- Partitioning around mediods (PAM): instead of averages, use multidim medians as centroids (cluster “prototypes”). Dudoit and Freedland (2002).
- Self-organizing maps (SOM): add an underlying “*topology*” (neighboring structure on a lattice) that relates cluster centroids to one another. Kohonen (1997), Tamayo et al. (1999).
- Fuzzy k-means: allow for a “gradation” of points between clusters; *soft partitions*. Gash and Eisen (2002).
- Mixture-based clustering: implemented through an EM (Expectation-Maximization) algorithm. This provides *soft partitioning*, and allows for *modeling* of cluster centroids and shapes. Yeung et al. (2001), McLachlan et al. (2002)

Evaluating a clustering solution:

Besides dendrogram cut height, or final value of the total within cluster sum of squares, a clustering can be evaluated through **Silhouette widths**. For each point $i=1 \dots N$ define:

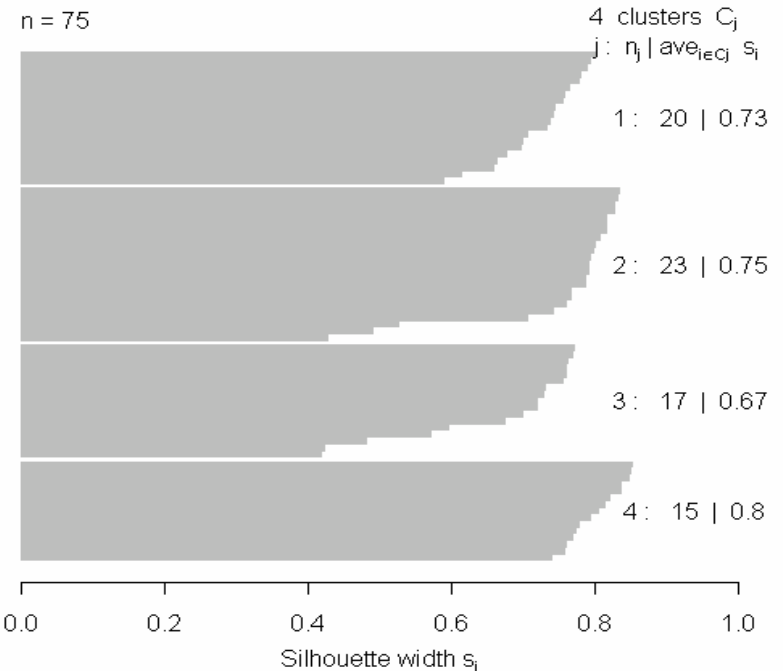
$$\delta(i, C) = \frac{1}{\#(C)} \sum_{j \in C} d(x_i, x_j)$$

$$a_i = \delta(i, C(i)) \quad C(i) = C \text{ st } i \in C$$

$$b_i = \min_{C \neq C(i)} \delta(i, C)$$

$$sil_i = \frac{b_i - a_i}{\max\{a_i, b_i\}}$$

Silhouette plot



Average silhouette width : 0.74

Silhouette of 75 units in 4 clusters:						
Cluster sizes and average silhouette widths:						
	20	23	17	15		
	0.7262347	0.7548344	0.6691154	0.8042285		
Individual silhouette widths:						
	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
	0.4196	0.7145	0.7642	0.7377	0.7984	0.8549

Observations with large and positive sil (~ 1) are well clustered, those with sil around 0 lie between clusters, and those with negative sil are placed in the “wrong” cluster.

Summary statistics for sil_i , $i=1 \dots N$; plots of them (most commonly, by cluster).

Computational assessments:

Because methods are so sensitive to possibly “anomalous” positions of points, and distributional assumptions hard to make, computational assessments (stability, bootstrapping) are also very important:

- Perturb adding noise (drawn from what distribution?) to the data.
- Perturb deleting points (resample without replacement) from the data.
- Bootstrap (resample with replacement) the data.

Is the clustering solution (dendrogram and chosen partition; partition in K groups and choice of K) stable to perturbations? What is their sampling variability?

We will explore these more in relation to choice of K (where to cut the dendrogram; how many groups to postulate).