Clustering methods:

Identify groups of observations that are similar with respect to a certain number of variables. Similarity is "metric": proximity, given some distance, in a space where observations are points, and dimensions are the variables recorded on the observations.

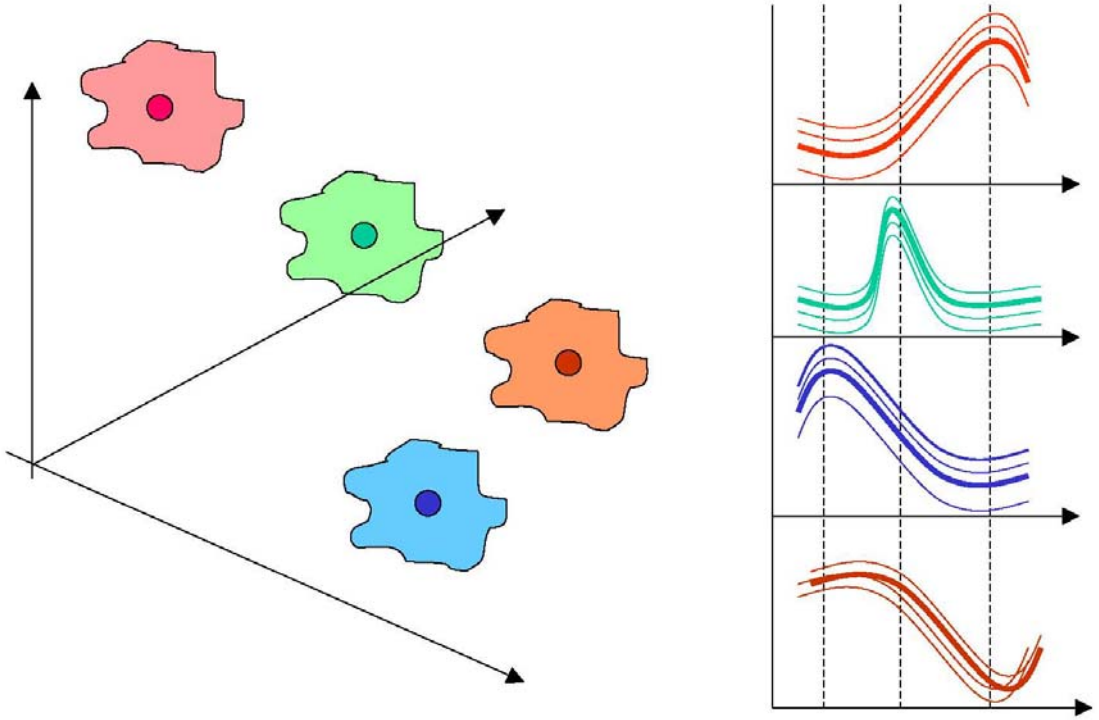
In microarray data analysis, it may be useful to cluster:

• "Conditions", when they correspond to samples. In this case we have T observations and N variables (*T* points in an *N*-dim space). Unsupervised classification, or "class discovery" for the samples, based on gene expression.

• Genes. In this case we have N observations *T* variables (*N* points in a T-dim space). ***Partition genes*** into classes presenting similar expression profiles over the conditions of interest. Rationale: genes with similar expression profiles may be involved in similar/related functions, and possibly be co-regulated. But the discussion on this is wide open… Also, ***extract "characteristic expression patterns"*** as cluster centroids.

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

Basic cartoon:



Pre-processing of the data matters:

• Does it make sense to center and/or standardize by row? (then a typical Euclidean distance will score as similar profiles with similar shapes, regardless of their overall level or size).
• Does it make sense to center and/or standardize by column? (then a typical Euclidean distance will be insensitive to original differences in variability scales and sizes for different “conditions”)
• In general, does it make sense to apply any left or right affine transformation to our data matrix $\boldsymbol{X}$?

These questions may be posed equivalently in terms of choice of distance (other than Euclidean), and they are core to clustering algorithms.

The literature on clustering is huge (developments related to data mining); our discussion will be far from a complete picture. We consider only:

• Algorithms that create partitions of the data (**Exclusive**; overlapping clusters are not allowed; algorithms allowing for overlapping do exist).
• Algorithms that do not use additional information on the possible groups (**Unsupervised**; intrinsic) – Supervised algorithms go under “classification”; working with a response.

Hierarchical Clustering (agglomerative)

Set of *N* data points in R^T. Define similarity/dissimilarity; choose a distance in the space, or give directly a *NxN* matrix of distances between pairs of points. Proceeding in an agglomerative fashion ("bottom-up"), generate a sequence of nested partitions of the data – progressively less fine – starting with *N* clusters (each containing a single point), and ending with one cluster (containing *N* points).

- 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 *NxN* matrix.
- Choose a distance function for clusters, $D(C_1,C_2)$, based on a summary of the distances among points, as measured by $d(x_1,x_2)$ (for clusters formed by just one points, *D* reduces to *d*).
- Start from *N* clusters, each containing one data point.
- At each iteration, proceed as follows:

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.

• The method is very sensitive to anomalous data points/outliers
• Merging is un-revocable (cannot split a cluster after having created it): a “bad” merging occurring early on will be carried all the way down, affecting the structure of the nested sequence.
• 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 method

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

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 the distance *d* – not the case for average linkage. However, the latter is often preferred as a compromise because single linkage tends to produce “long”, “stringy” clusters (single) and “small”, “compact” clusters (complete).

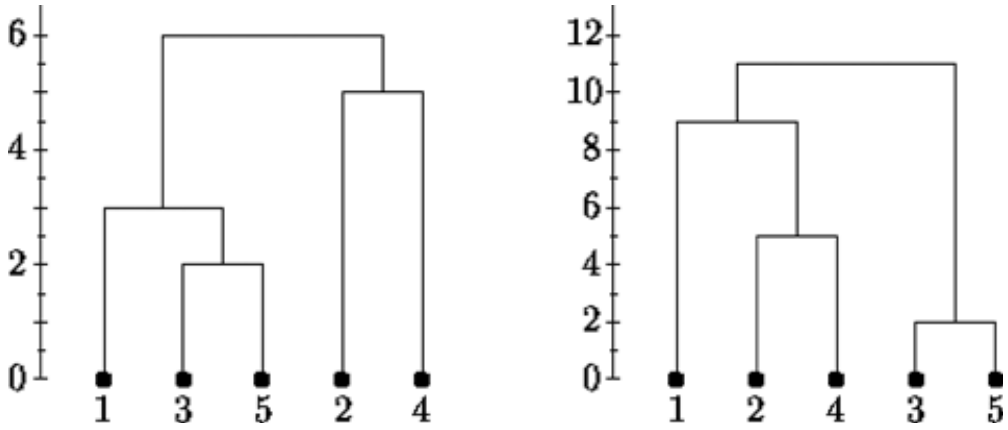
Example

Agglomeration step (first) in constructing the nested sequence (first iteration): On the left is the matrix of distances among 5 data points. 3 and 5 are the closest, and are therefore merged in cluster “35”. On the right is the 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). The ordinate shows the distance at which the merging occurred. The horizontal ordering of the data points is any order preventing intersections of dendrogram branches.



Hierarchical clustering, per se, does not dictate a partition and a number of clusters. It provides a nested sequence of partitions, each containing one less cluster. This is more informative than just a partition. To settle on one partition, we have to decide where to "cut" the dendrogram. We may do so based on

- A threshold distance (dissimilarity) level we are willing to "tolerate" within a cluster
- Looking for obvious "leaps" in distance as we move along the dendrogram
- Looking for an obvious "bend" in distance as we move along the dendrogram

In fact, if we are fairly confident that the selected linkage method is appropriate, and "accidents" along the merging sequence are minor, we could read the distance level associated to various numbers of clusters *D(K,link,seq)=D(K)*, from the top of the dendrogram (*K*=1 clusters) to the bottom (*K*=*N*), as a measure of "fit" of the partitioning in K groups.

Hierarchical clustering of both "rows" (called observations) and "columns" (called variables), is easily performed in **Minitab**.

- Various options for distance and linkage choice.
- Can produce the dendrogram (NOT for thousands of genes).
- Can specify a number of clusters or a similarity level to settle on one partition along the nested sequence (possibly one cluster; will go all the way up).

Corresponding **cluster memberships** *m(x)* are produced for each observation.

Partitioning algorithms: K-means.

Again, need to select a distance functions for points. Here, also need to fix the desired number of clusters in the output partition; *K* (<*N*). Starting from given initial locations of the *K* cluster centroids, the algorithm uses the data points to iteratively relocate the centroids, and reallocate points to the closest centroid.

- Choose a distance function for points, $d(x_1,x_2)$.
- Choose a *K*.
- Initialize the *K* centroids (e.g. *K* data points at random) $\bar{x}_1(0),\ldots\bar{x}_K(0)$
- At each iteration, proceed as follows:

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

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

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

$$m(x)=\arg\min_{k=1...K} d(x,\bar{x}_k) \quad \forall x$$

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

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

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

• Also this method is very sensitive to anomalous data points/outliers.
• Points can “move” from one cluster to another as the algorithm proceeds.
• If two cluster centroids are equally (and maximally) close to an observation at a given iteration, we have to choose arbitrarily to what cluster to attribute the point (the problem here is not so serious because points can move…)
• There are several “variants” of this algorithm (e.g. update only the centroid of the cluster getting a new observation).

The k-means algorithm is guaranteed to converge to a local minimum of the **total within-cluster square distance** (with Euclidean distance, this is a total within cluster sum of squares), but not necessarily to a global one.

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

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

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

• The second form uses only the centroids as “optimization variables” (memberships can be derived from them; helps us see how final value of the objective function depends on centroid initialization).
• The third form is useful in comparison with SOM and fuzzy k-means.

Other relevant remarks:

• A reasonable *K* may be unknown
• Initialization of the centroids may make a difference (determine in what local minimum we end up)
• Are cluster averages the most representative "prototypes" for clusters?
• Cluster boundaries tend to have the shape of a "ball" wrt the chosen distance.

If we are fairly confident that initialization allows us to reach a "good" minimum, we could use $\Delta(K,start)=\Delta(K)$ as a measure of "fit" of the partitioning in *K* groups and calculate it for several values of K. We could then fix a threshold, or look for "leaps" or "bends" in $\Delta(K)$, to select a *K*.

K-means clustering of "rows" (observations) on the basis of "columns" (variables), is easily performed in **Minitab** (Euclidean distance only).

Can specify the final partition through a choice of *K*, or of an initial partition of the points. Corresponding **cluster memberships** *m(x)* are produced for each observation.

Variants:

• 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).

Also often used Mixture-based clustering, implemented through EM algorithm. This provides *soft partitioning*, and allows for *modeling* of cluster centroids and shapes. Yeung et al. (2001), McLachlan et al. (2002)

Some remarks for both hierarchical and K-means clustering

Because methods are so sensitive to possibly "anomalous" positions of points, and distributional assumptions hard to make, computational analyses (stability, boot-strapping) are 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

Are the outcomes (dendrogram and chosen partition; partition in *K* groups and choice of *K*) stable to perturbations? What is their sampling variability?

We will go into this for the choice of *K* (where to cut the dendrogram; how many groups to postulate).

How strong is the "association" of an observation *x* to its cluster, the *m(x)*-th? Cluster memberships do not provide this information, but we could compute

$$D(x,C_k) \quad \text{or} \quad d(x,\bar{x}_k)$$

for all clusters in the partition, to see if and by how much distance from the *m(x)*-th cluster is smaller than the other ones.