CSC 461: Machine Learning

Fall 2024

Hierarchical Clustering

Prof. Marco Alvarez, Computer Science
University of Rhode Island

Hierarchical clustering

- Unsupervised learning technique that groups observations into nested clusters
- Key features:
  - does not require a predefined number of clusters
  - creates a hierarchy of clusters, often visualized as a dendrogram
    - can be agglomerative (bottom-up), or divisive (top-down)
  - widely used in various domains (gene expression, phylogenetic trees, community detection, etc.)

Agglomerative vs divisive

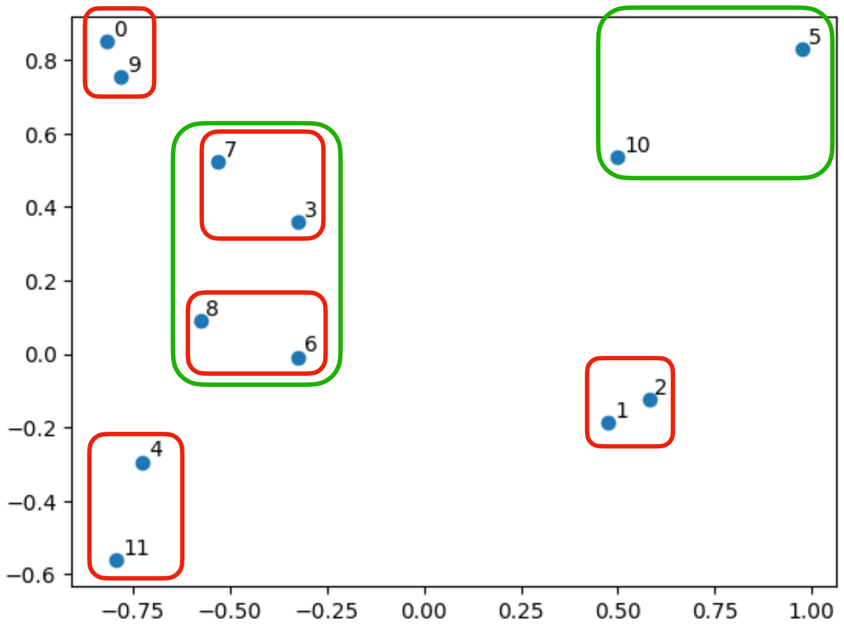
- Agglomerative approach (**bottom-up**)
  - start with observations as singleton clusters and progressively merge them until a final single cluster is created
  - most common form
- Divisive approach (**top-down**)
  - start with all observations on a single cluster and progressively split the clusters until all observations are singleton clusters

HAC algorithm

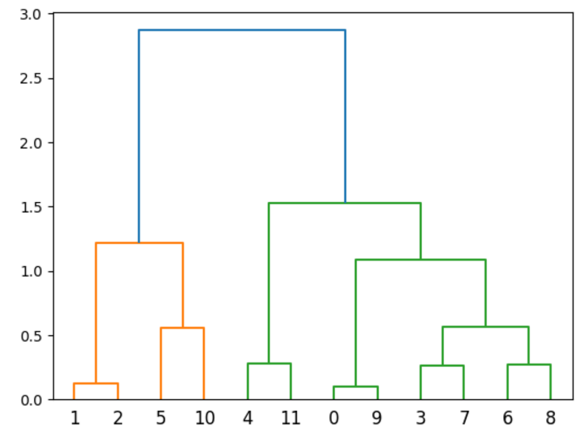
- Initially each observation is considered a cluster
- Repeat
  - **merge:** find the closest pair of clusters and merge it into a single cluster
  - if all observations are in a single cluster stop

```
Distance Metrics:
 – Euclidean distance
 – Manhattan distance
 – Minkowski distance
 – Hamming distance
 – Cosine similarity
 – Correlation coefficient

Linkage Criteria:
 – Single–linkage
 – Complete–linkage
 – Average–linkage
 – Centroid–linkage
 – Ward's method
```



Dendrogram

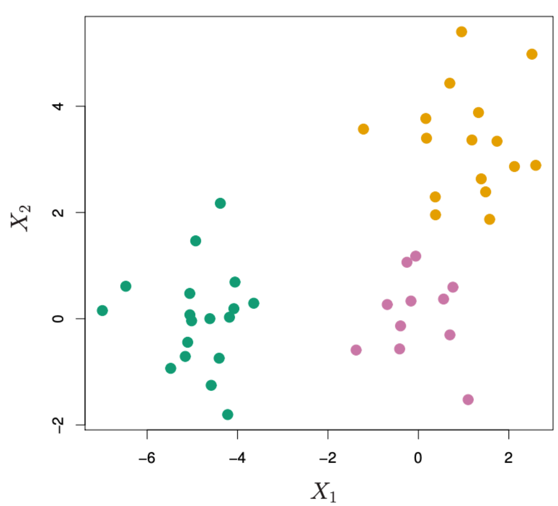


A dendrogram represents the hierarchy of clusters

Y-axis represents the distance at which clusters are merged

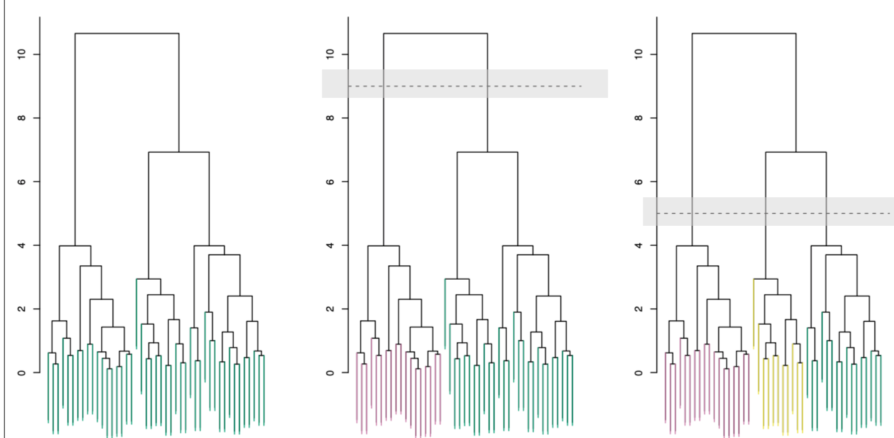
Horizontal cuts through the dendrogram represent cluster groupings at various thresholds

Consider this data



An Introduction to Statistical Learning, 2nd Ed

HAC with complete-linkage



An Introduction to Statistical Learning, 2nd Ed

Pairwise metrics

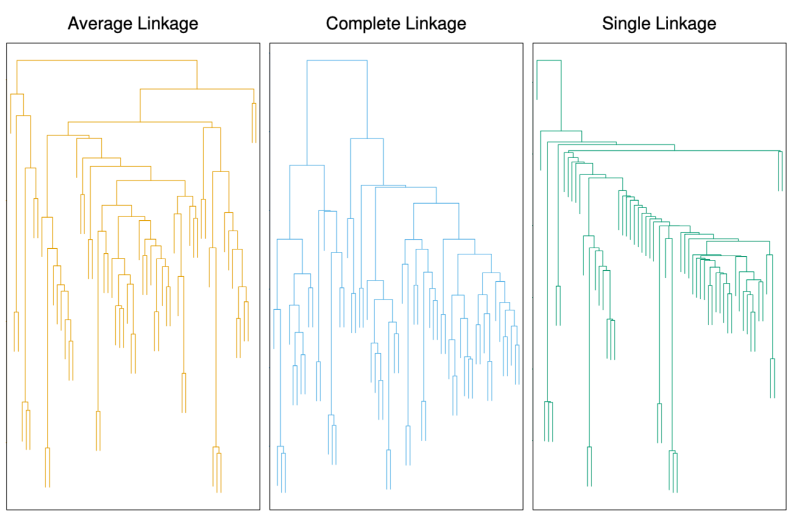
See the Pairwise metrics, Affinities and Kernels section of the user guide for further details.

`metrics.pairwise.additive_chi2_kernel`(X[, Y])	Compute the additive chi-squared kernel between observations in X and Y.
`metrics.pairwise.chi2_kernel`(X[, Y, gamma])	Compute the exponential chi-squared kernel between X and Y.
`metrics.pairwise.cosine_similarity`(X[, Y, ...])	Compute cosine similarity between samples in X and Y.
`metrics.pairwise.cosine_distances`(X[, Y])	Compute cosine distance between samples in X and Y.
`metrics.pairwise.distance_metrics`()	Valid metrics for pairwise_distances.
`metrics.pairwise.euclidean_distances`(X[, Y, ...])	Compute the distance matrix between each pair from a vector array X and Y.
`metrics.pairwise.haversine_distances`(X[, Y])	Compute the Haversine distance between samples in X and Y.
`metrics.pairwise.kernel_metrics`()	Valid metrics for pairwise_kernels.
`metrics.pairwise.laplacian_kernel`(X[, Y, gamma])	Compute the laplacian kernel between X and Y.
`metrics.pairwise.linear_kernel`(X[, Y, ...])	Compute the linear kernel between X and Y.
`metrics.pairwise.manhattan_distances`(X[, Y, ...])	Compute the L1 distances between the vectors in X and Y.
`metrics.pairwise.nan_euclidean_distances`(X)	Calculate the euclidean distances in the presence of missing values.
`metrics.pairwise.pairwise_kernels`(X[, Y, ...])	Compute the kernel between arrays X and optional array Y.
`metrics.pairwise.polynomial_kernel`(X[, Y, ...])	Compute the polynomial kernel between X and Y.
`metrics.pairwise.rbf_kernel`(X[, Y, gamma])	Compute the rbf (gaussian) kernel between X and Y.
`metrics.pairwise.sigmoid_kernel`(X[, Y, ...])	Compute the sigmoid kernel between X and Y.
`metrics.pairwise.paired_euclidean_distances`(X, Y)	Compute the paired euclidean distances between X and Y.
`metrics.pairwise.paired_manhattan_distances`(X, Y)	Compute the paired L1 distances between X and Y.
`metrics.pairwise.paired_cosine_distances`(X, Y)	Compute the paired cosine distances between X and Y.
`metrics.pairwise.paired_distances`(X, Y, *[, ...])	Compute the paired distances between X and Y.
`metrics.pairwise_distances`(X[, Y, metric, ...])	Compute the distance matrix from a vector array X and optional Y.
`metrics.pairwise_distances_argmin`(X, Y, *[, ...])	Compute minimum distances between one point and a set of points.
`metrics.pairwise_distances_argmin_min`(X, Y, *)	Compute minimum distances between one point and a set of points.
`metrics.pairwise_distances_chunked`(X[, Y, ...])	Generate a distance matrix chunk by chunk with optional reduction.

Distances between clusters

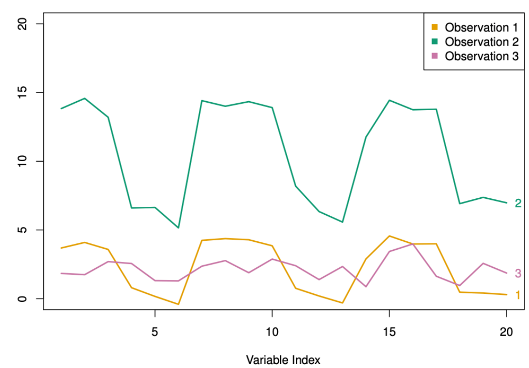
Single-linkage:	$D(A,B) = \underset{\mathbf{x}_1 \in A, \mathbf{x}_2 \in B}{\arg\min}\ d(\mathbf{x}_1, \mathbf{x}_2)$
Complete-linkage:	$D(A,B) = \underset{\mathbf{x}_1 \in A, \mathbf{x}_2 \in B}{\arg\max}\ d(\mathbf{x}_1, \mathbf{x}_2)$
Average-linkage:	$D(A,B) = \frac{1}{\lvert A \rvert \lvert B \rvert} \sum_{\mathbf{x}_1 \in A, \mathbf{x}_2 \in B} d(\mathbf{x}_1, \mathbf{x}_2)$
Centroid-linkage:	$D(A,B) = \lVert \mu_A - \mu_B \rVert_2^2$
Ward's method:	Minimizes variance within clusters

Example



What kind of distance to consider?

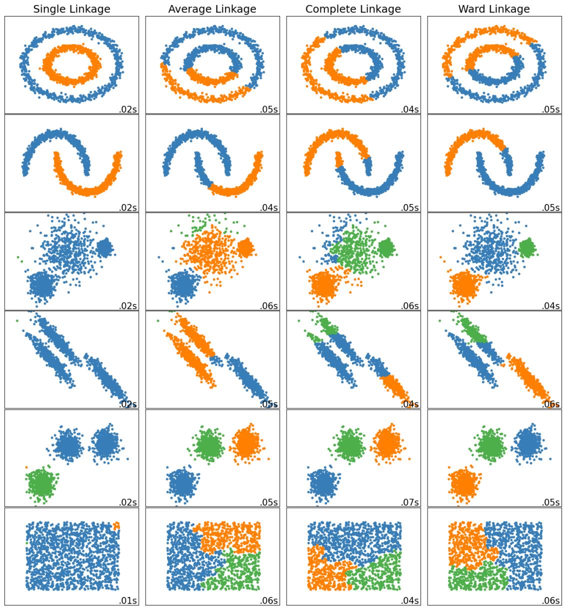
- We have been using Euclidean distances, however, the choice is **very important**



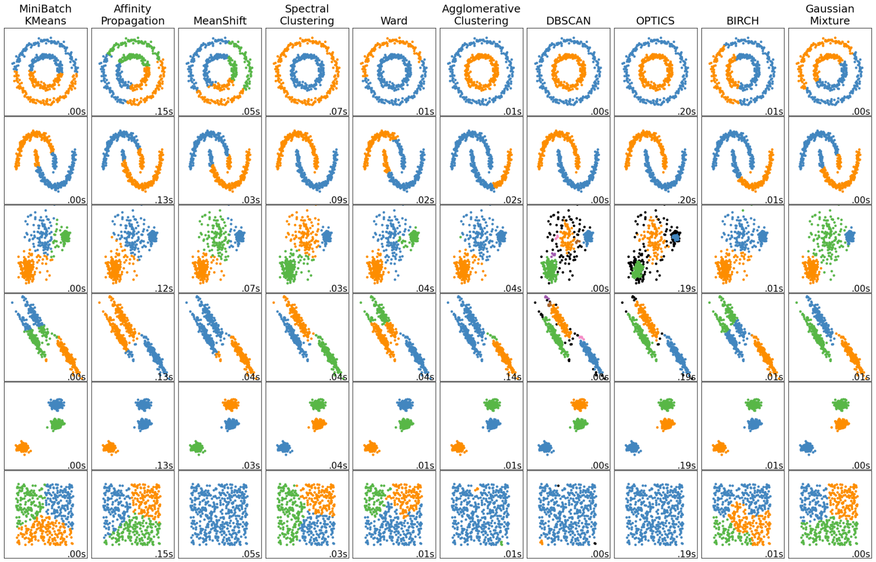
Observations 1 and 3 have a small Euclidean distance between them. But they have a large correlation-based distance. Observations 1 and 2 have a large Euclidean distance between them. But they have a small correlation-based distance.

Limitations and considerations

- Computational complexity
  - can be computationally expensive for large datasets
- Sensitivity to noise and outliers
  - can be sensitive to noise and outliers in the data
- Choice of distance metric and linkage criterion
  - distance metric and linkage criterion can significantly affect the results
- Optimal number of clusters
  - no definitive method for determining the optimal number of clusters



From scikit-learn



From scikit-learn