Lecture 12:

Unsupervised Learning: Clustering

Week of February 20, 2023

University California, Berkeley

Machine Learning Algorithms

MSSE 277B, 3 Units

Spring 2023

Prof. Teresa Head-Gordon

Departments of Chemistry, Bioengineering, Chemical and Biomolecular Engineering

Summary of Previous Lecture

Maximum Entropy is a technique for learning probability distributions from data to predict $P(y_k|x_1, \ldots x_p)$ directly; it is considered a discriminative model. It is based on following principles and features:

- Unlike Naive Bayes classifier, Maximum Entropy does not assume that {x} data are conditionally independent of each other.
- Equivalent to logistic regression which does not assume independence in features!
- The Principle of Maximum Entropy selects model that fits the training data with the most uniform distribution subject to the observed data constraints. This is equivalent to maximizing "energy dispersal" under constraints of NVT and p behaving as probability in statistical mechanics!
- Also known as SoftMax (vector of real numbers to vector of numbers that sum to one.)
- We can use cross-entropy as a cost function to train in classification for many categories

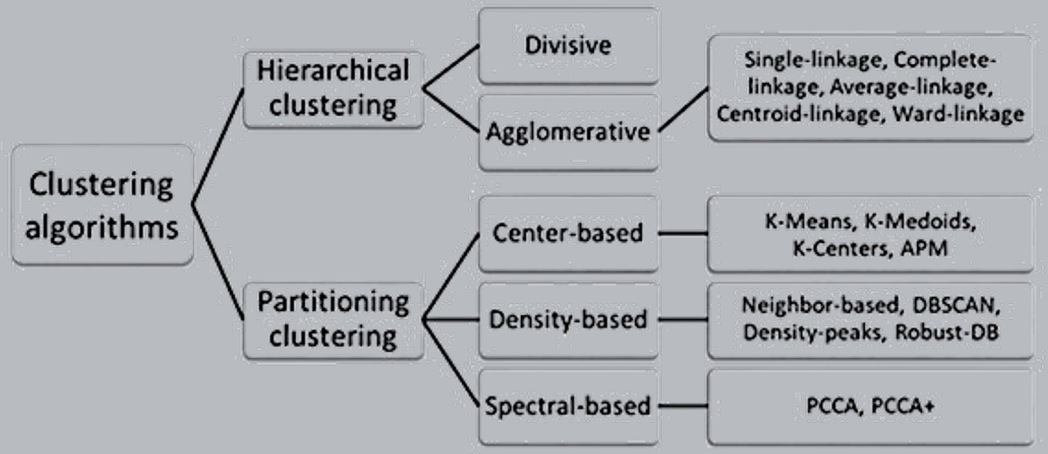
Unsupervised Learning

Purpose of Today's Lecture: In unsupervised learning, no Y labels are given to the learning algorithm, and we thus find structure in the {x} inputs themselves.

Unsupervised learning can be a goal in itself (discovering hidden patterns in data) or a means towards an end (feature learning for a discriminative/generative statistical model or for machine learning).

The goal in unsupervised learning problems may be to (1) discover groups of similar examples (clustering), or (2) to determine how the data is distributed in the space, known as density estimation.

We consider both non-parametric (assumes no information about the cluster shape, distribution etc) vs. parametric, and weigh up pros and cons of each method.

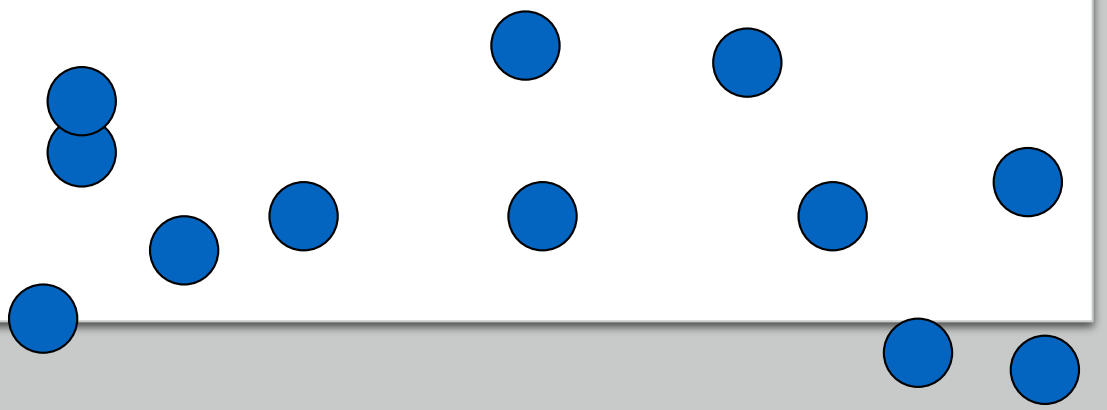


Unsupervised Learning: Clustering Data

Non-parametric Unsupervised Learning groups data into clusters, where each cluster hopefully is interpretable about categories and classification outcomes present in the data.

Nonparametric are sometimes referred to as distribution-free methods. One example is K-Means clustering. K-means is the most widely-used clustering algorithm due to its efficiency, and hence we start there!

Let $X = \{x_1, \dots x_N\}$ be the set of data points. The goal is to organize data into K clusters with centers, $C = \{c_1, \dots c_K\}$, by minimizing a loss function.



Loss Functions in Clustering

One loss function is the Euclidean distance $\boldsymbol{d}$

$$loss = \sum_{j}^{K} \sum_{i \in C_j}^{N} d(\vec{x}_i, \vec{c}_j)^2 = \sum_{j}^{K} \sum_{i \in C_j}^{N} (\vec{x}_i - \vec{c}_j)^2$$

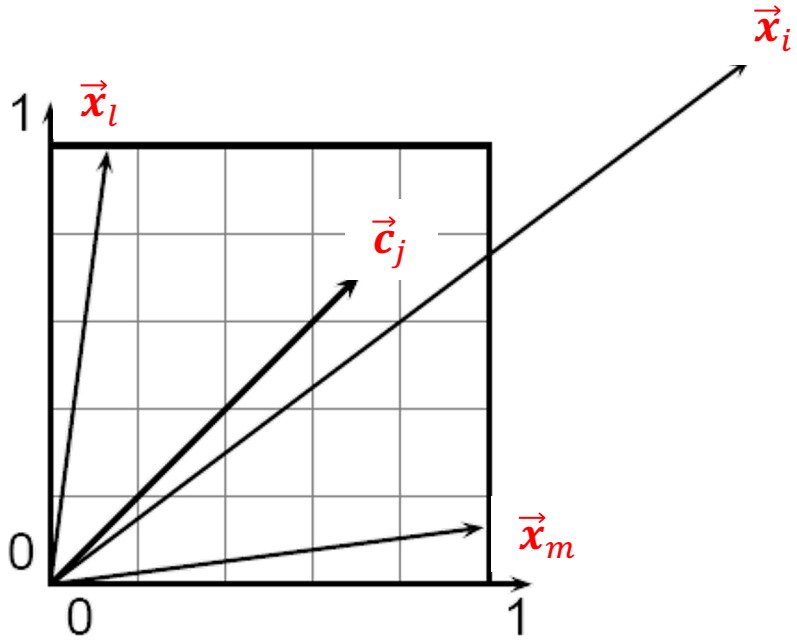
Resulting clusters are invariant to rotation and translation of data, but not to scaling. If features have different scales, it may be best to normalize data – but maybe not (as different scales are part of the distinguishing feature!)

Another metric is Cosine Distance

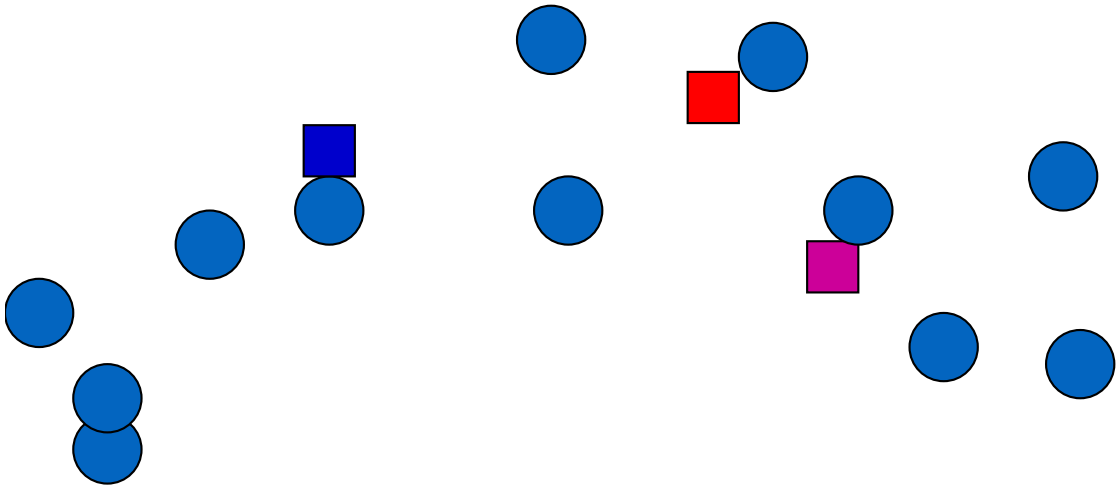
$$\boldsymbol{cos}(\vec{x}_i, \vec{c}_j) = \frac{\vec{x}_i \cdot \vec{c}_j}{\|\vec{x}_i\| \|\vec{c}_j\|}$$

Note that "similarity" can change!

- By Euclidean distance $\vec{x}_i$ *and* $\vec{c}_j$ are more different than other data.
- Under Cos distance, $\vec{x}_i$ *and* $\vec{c}_j$ are most similar

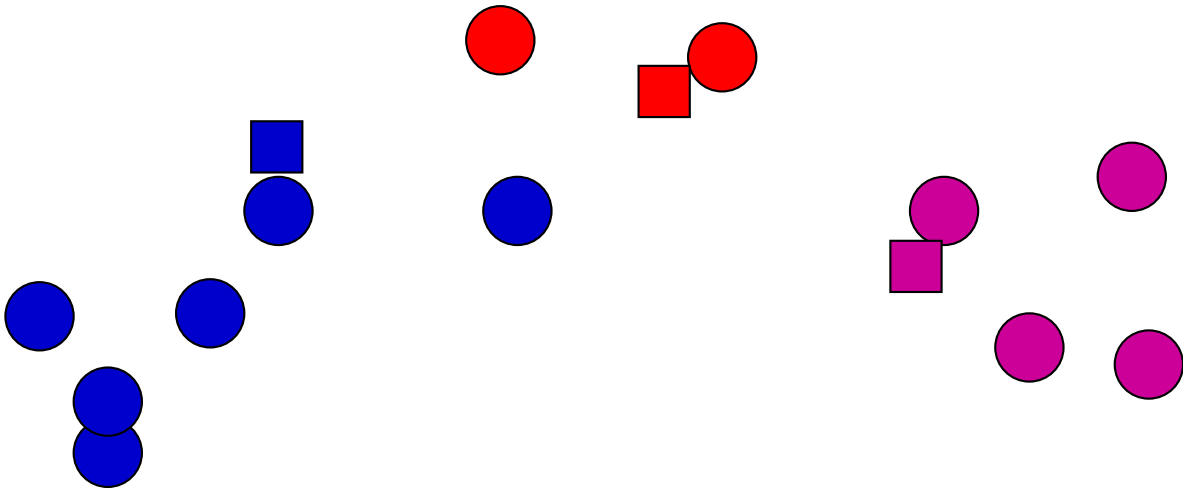


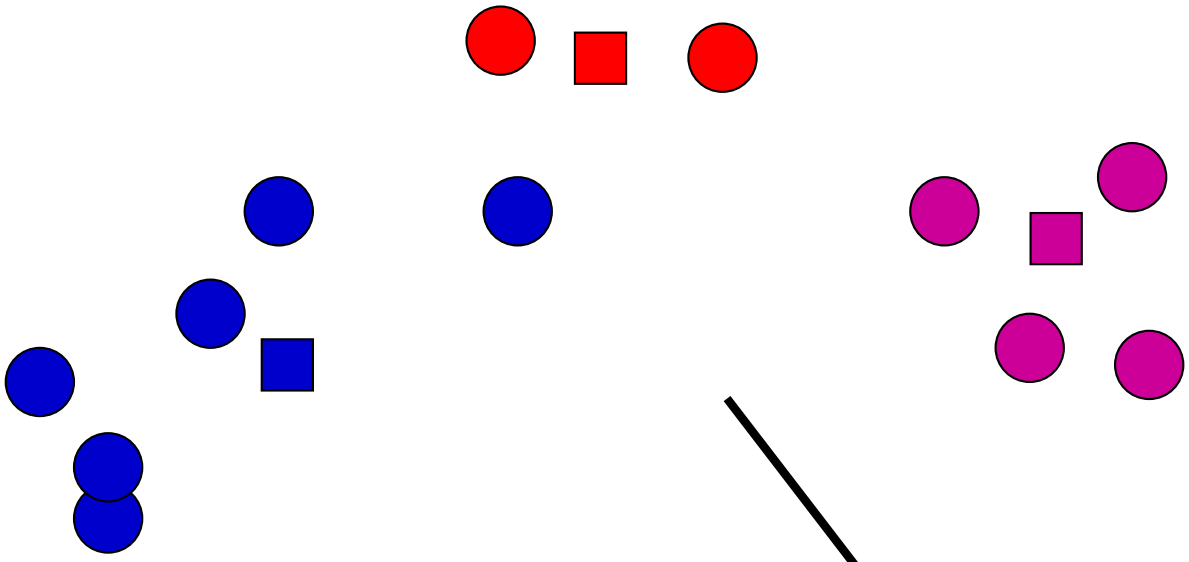
K-Means Clustering: Most popular!



(1) For K-Means we initiate by randomly defining the C cluster centers (squares). Euclidean distance is used (need advanced K-Means to do other loss functions)

(2) Calculate distance between each x_i, c_j. Assign data point to cluster whose distance from cluster center is the minimum compared to all of the other cluster centers.

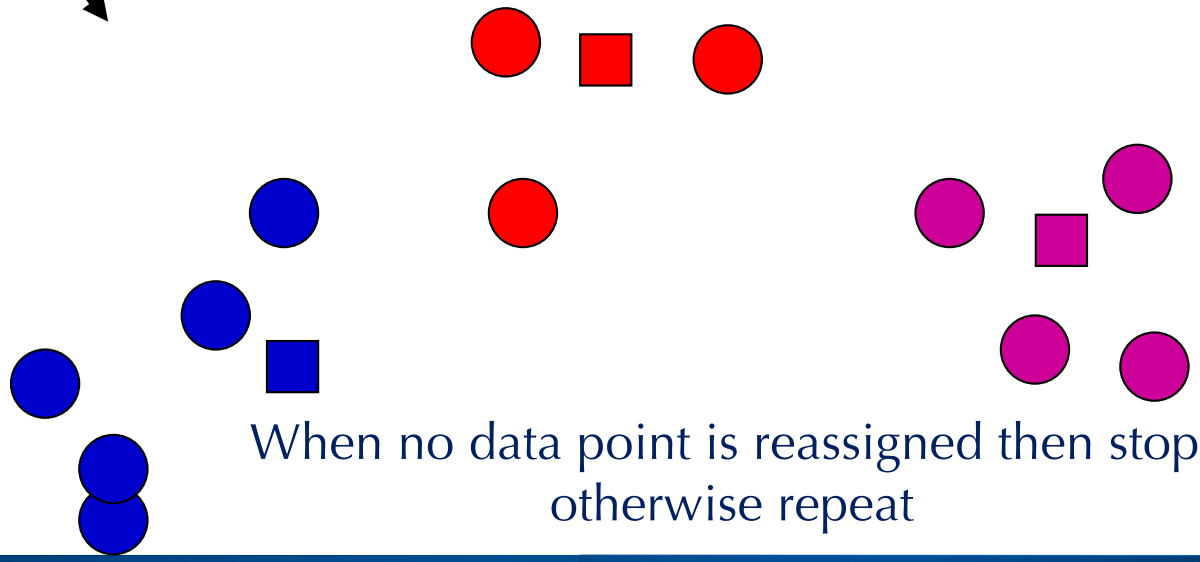




(3) Recalculate new cluster centers as mean of the points in a cluster

$$\vec{c}_j = \frac{1}{N_j} \sum_{i \in C_j}^{N} \vec{x}_i$$

(4) Recalculate distance between each data point and obtain new cluster assignments.

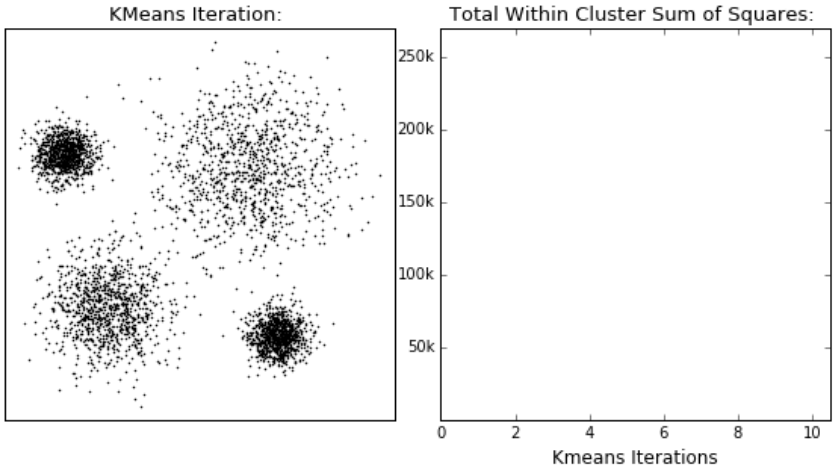


When no data point is reassigned then stop, otherwise repeat

K-Means Clustering

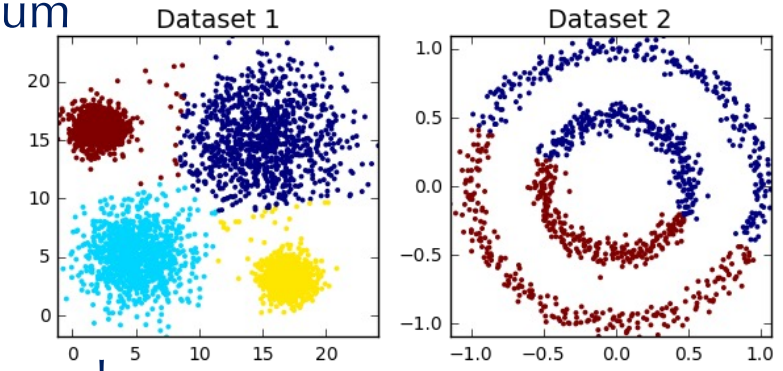
PROs

- Easy to implement (its best feature!)
- Data points adapt and change cluster when the c computed.
- K-Means may be computationally faster than other clustering methods (if K is small)
- Easy to interpret



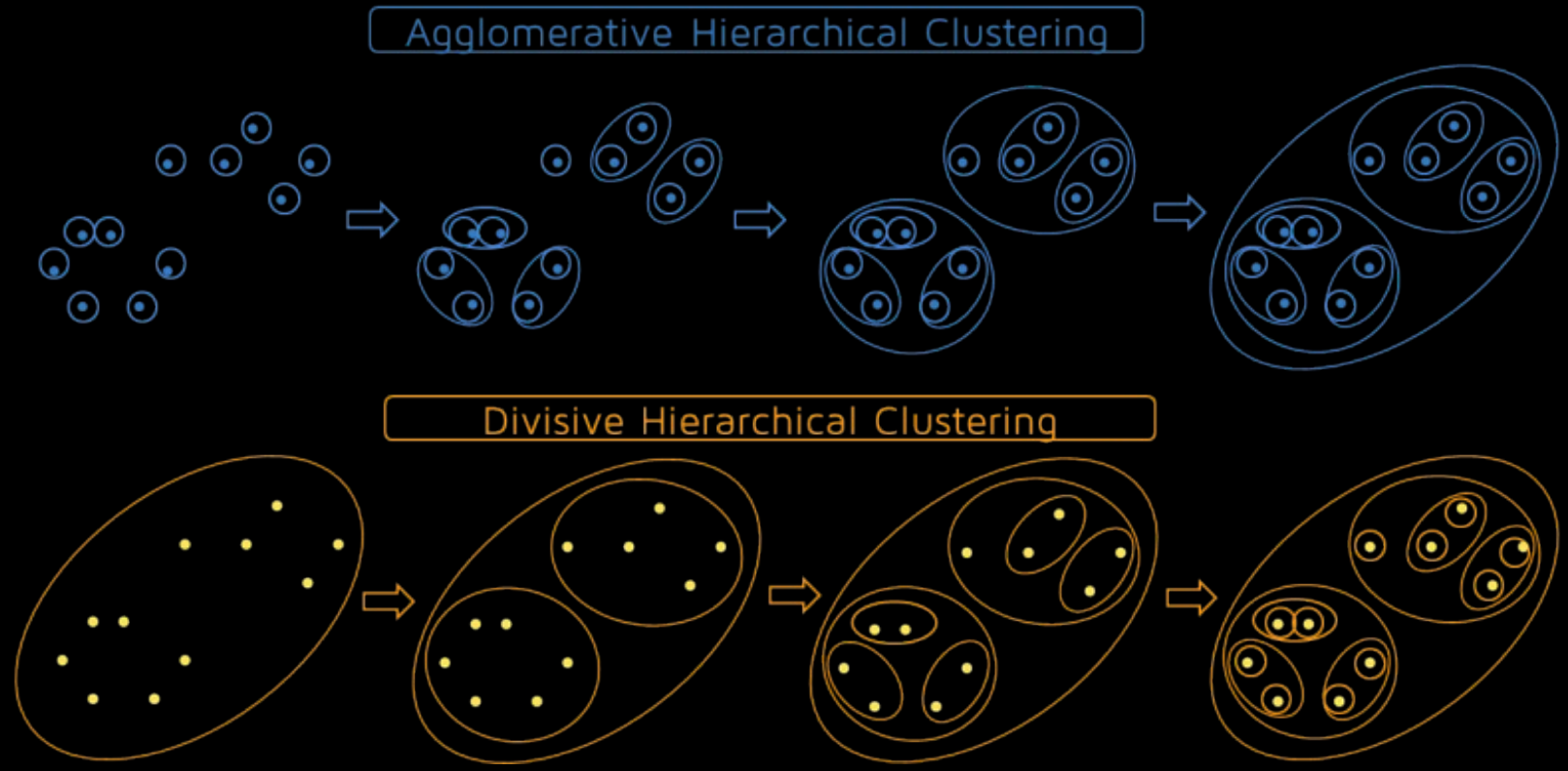
CONS

- Guaranteed to find a minimum, but not global minimum
- Sensitive to initial conditions. K-means should be run multiple times to reduce this effect
- If no samples closest to c_j, cannot be updated.
- No optimal "K". Increasing K results in smaller error but can increase risk of overfitting
- Not suitable for discovering clusters that are not hyper-spheres.



K-Means Clustering: Pros

Divisive (top down) clustering and Agglomerative (bottom up) clustering. Built from top (bottom) level by splitting (merging) the most dissimilar (similar) pair of clusters. Stopping when all the data points are merged into individual points (single root cluster).



Hierarchical Clustering

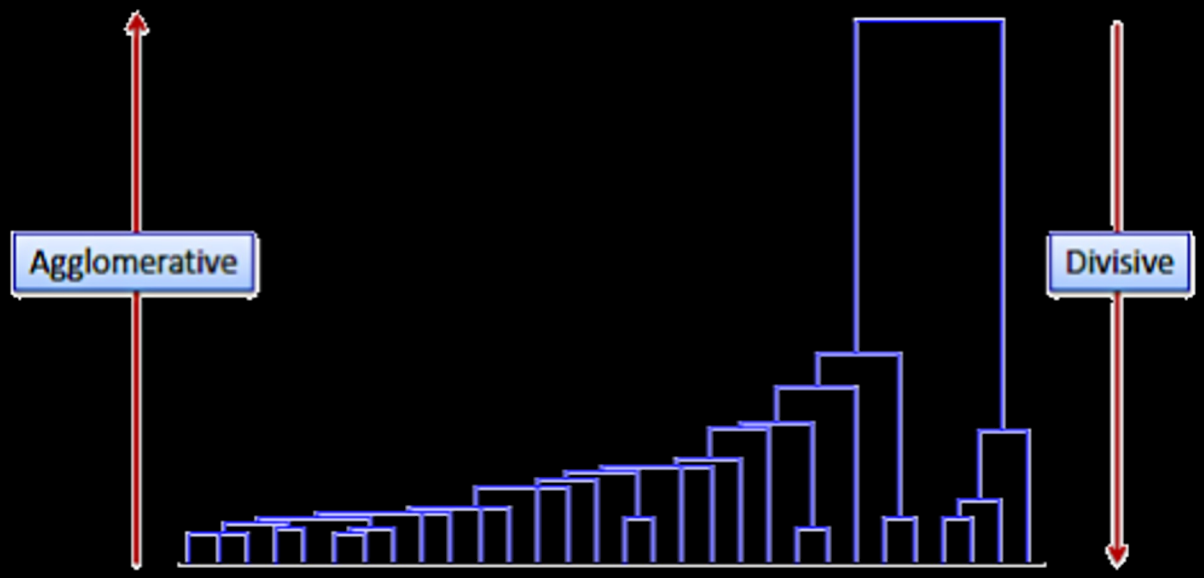
Hierarchical clustering does not require a pre-specified choice of K and provides a deterministic answer (unlike K-means).

Quantified as a Dendrogram:

- Vertical lines represent clusters that are joined together.
- Position of line on scale indicates distances at which clusters were joined.

Produces not one clustering, but a family of clusterings

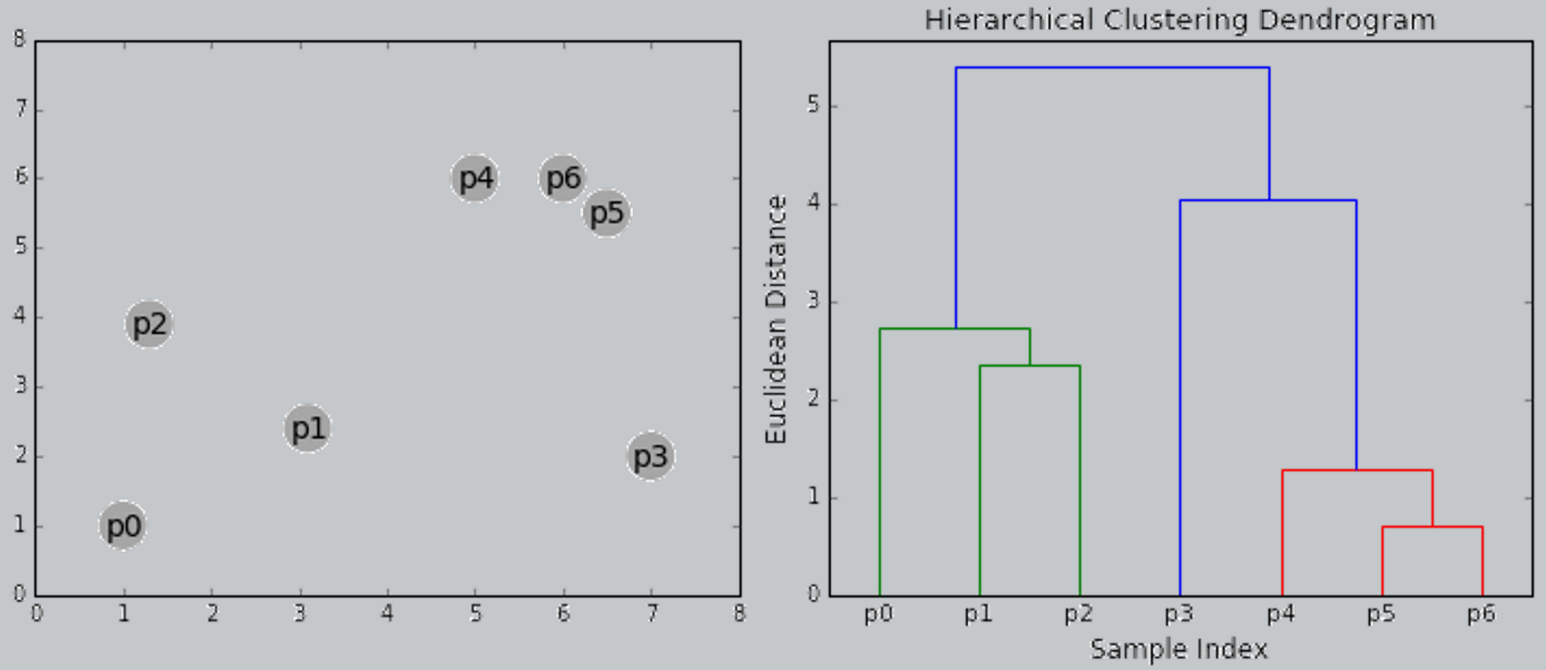
- The earlier fusion occurs, the more similar the observations are to each other
- The height of the fusion indicates how different the two observations are.
- We cannot draw conclusions about the similarity of two observations based on their proximity along the x axis



Hierarchical Clustering

Start with each point $\overrightarrow{x_i}$ in its own cluster. We then merge the two observations that have the lowest dissimilarity $d(\overrightarrow{x_i}, \overrightarrow{x_j})$ (or highest similarity!). Repeat until only one cluster remains.

Find 2 clusters (c1, c2) that are most similar and merge into one cluster



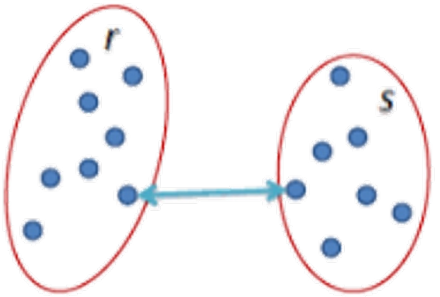
But we need to think about what dissimilarity means, not between points, but between clusters. The dissimilarity between two clusters is called the linkage, of which there are different types

Agglomerative Hierarchical Clustering

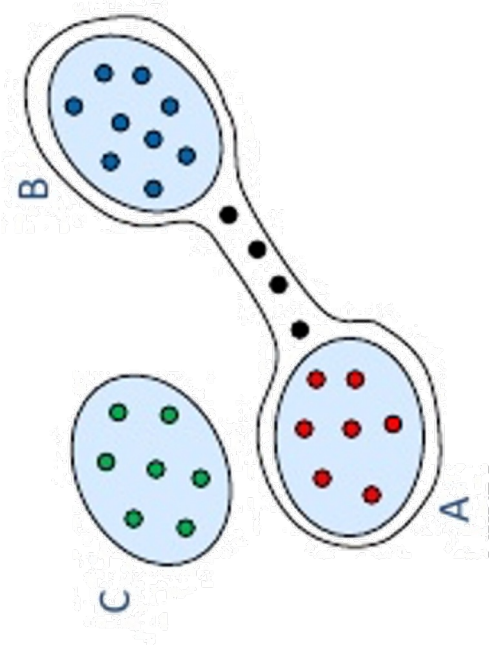
Hierarchical Clustering: Single Linkage

In single linkage, the distance between clusters is defined by the smallest dissimilarity between two points in different groups:

$$L_{single}(r,s) = i \in r, j \in s, \min d(\overrightarrow{x_i}, \overrightarrow{x_j})$$



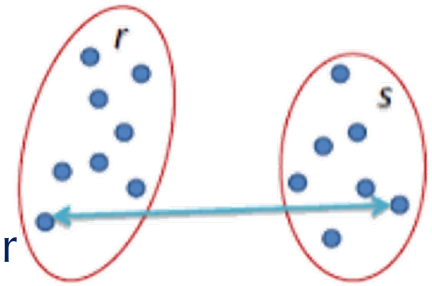
- Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and determine the smallest of these dissimilarities.



Single linkage can result in extended, chain-like clusters in which single datum are fused one-at-a-time

In complete linkage, the distance between clusters is defined by the largest dissimilarity between two points in different groups:

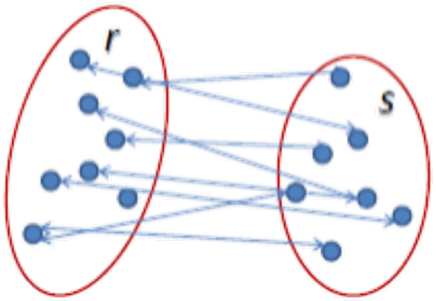
$$L_{complete}(r,s) = i \in r, j \in s, \max d(\vec{x_i}, \vec{x_j})$$



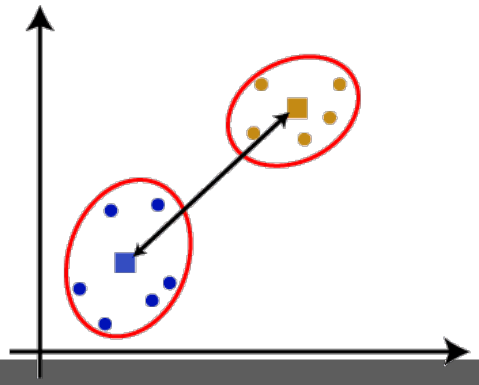
- Minimizes the problem of chaining seen in single linkage
- A drawback is points in one cluster may be closer to points in another cluster than to any of its own cluster: ″crowding″

In average or centroid linkage, the distance between clusters is average dissimilarity over all points in different groups:

$$L_{average}(r,s) = \frac{1}{N_r N_s} \sum_{i}^{N_r} \sum_{j}^{N_s} d(\vec{x_i}, \vec{x_j})$$



$$L_{centroid}(r,s) = d\left(\frac{1}{N_r}\sum_{i}^{N_r} \vec{x_i}, \frac{1}{N_s}\sum_{j}^{N_s} \vec{x_j}\right) = d(\langle \vec{x_r} \rangle, \langle \vec{x_s} \rangle)$$



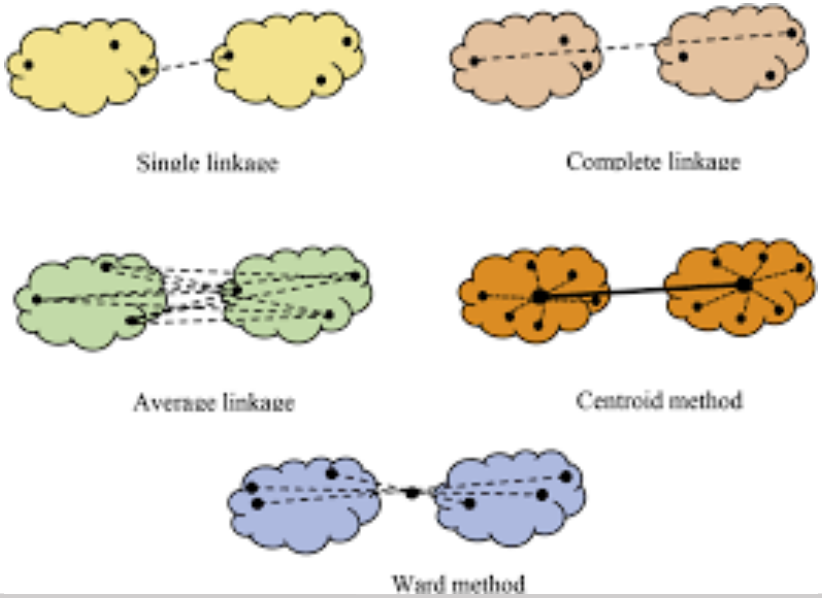
Best “compromise” between single and complete linkage

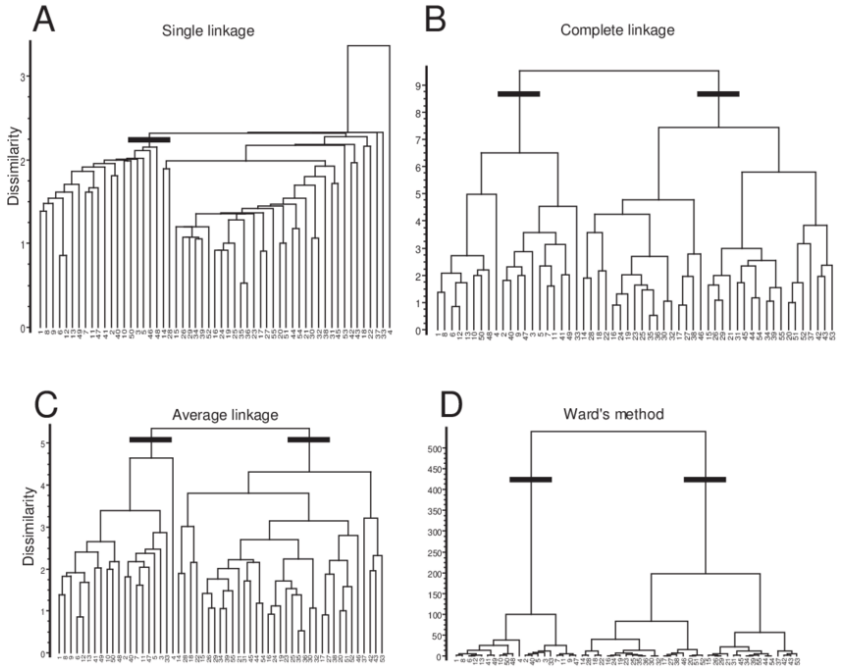
Hierarchical Clustering: Complete, Average, Centroid Linkage

Ward's minimum variance method: Ward's method is based on the objective of minimizing the deterioration in the overall within sum of squares. The sum of squares starts out at zero and then grows as we merge clusters.

Ward's method keeps this growth as small as possible.

$$L_{Ward}(r,s) = \sum_{k \in C_r \cup C_s}^{N} \left\| \vec{x}_k - \left\langle \langle \vec{x} \rangle_{C_r \cup C_s} \right\rangle \right\|^2 - \sum_{i \in C_r}^{N} \left\| \vec{x}_i - \left\langle \langle \vec{x} \rangle_{C_r} \right\rangle \right\|^2 - \sum_{j \in C_s}^{N} \left\| \vec{x}_j - \left\langle \langle \vec{x} \rangle_{C_s} \right\rangle \right\|^2$$

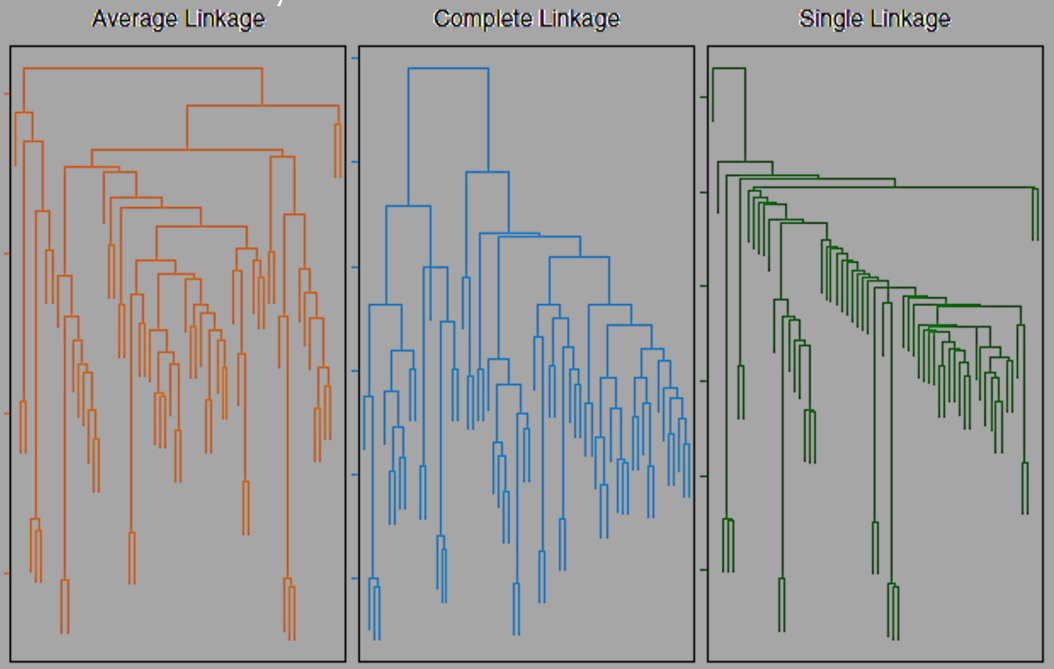




Hierarchical Clustering: Ward's Method

PROs

- Hierarchical clustering does not require a pre-specified choice of K and provides a deterministic answer (unlike K-means).
- It is easy to understand and easy to decide the number of clusters using Dendrogram.

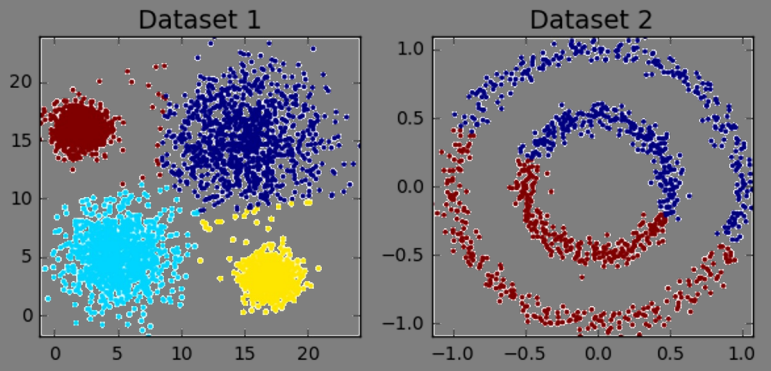


CONs

- All linkage approaches to calculate the (dis)similarity between clusters have their own disadvantages.
- Once a decision is made to combine two clusters, it can not be undone.
- Sensitivity to noise and outliers.

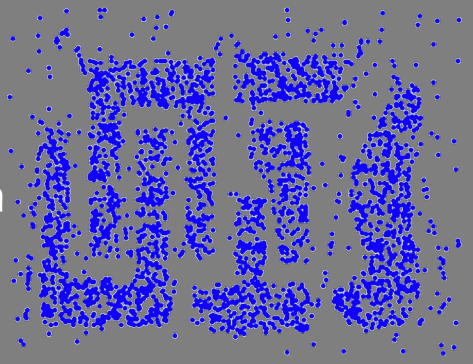
Hierarchical Clustering

Looking at density of data points is a common sense way to discover clusters in a dataset. K-Means and Hierarchical Clustering both fail in creating clusters based on density heterogeneity.

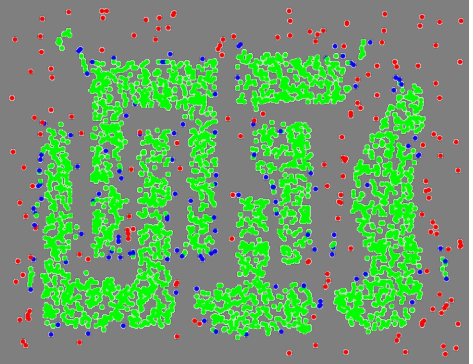


DBSCAN (Density-Based Spatial Clustering of Applications with Noise):

- Core Point: if it has more than a specified minimum number of points (MinPts) within some cutoff distance, Rcut. These points belong in a dense region
- Border Point: has fewer than MinPts within Rcut, but is in neighborhood of a core point.
- Noise point is any point that is not a core point or a border point.



Original Points



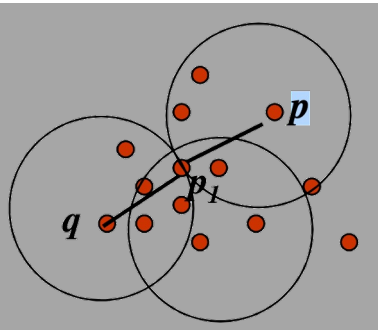
Point types: core, border and noise

Eps = 10, MinPts = 4

Density-Based Clustering

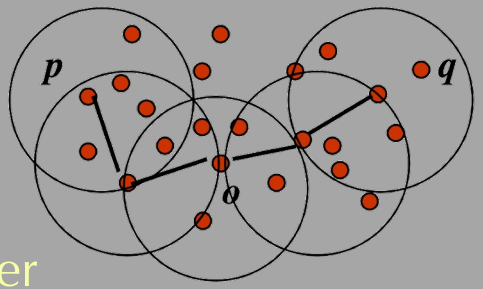
Density-connected points

- Density edge: We place an edge between two core points q and p if they are within distance Rcut.
- Density-connected: Any point p is density-connected to a core point q if there is a path of edges from p to q



DBSCAN algorithm:

Do Loop: pick a point, p, until all points have been visited
  mark p as visited
  if p neighbors < MinPts and Rcut> distance to core points then
    mark as noise point
  else if if p neighbors > MinPts and has not been assigned to a cluster
    create a new cluster with point p
    all points from p within Rcut are assigned in same cluster (mark them as visited)
    find all points that are density-connected to p and assign to cluster
  else assign border points to the cluster of the closest core point.
End do



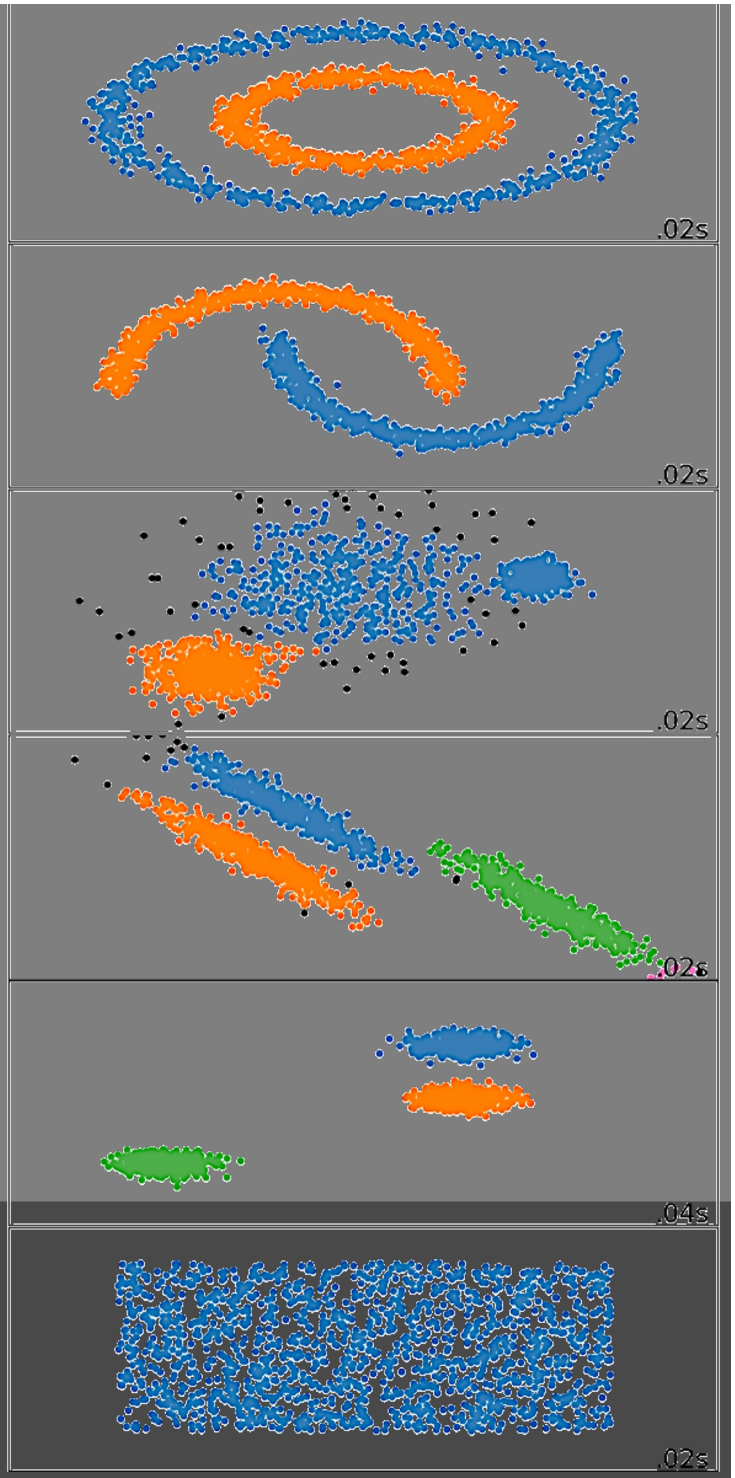
Density-Based Clustering: DBSCAN

PROs

- does not require one to specify the number of clusters
- performs well with arbitrary shaped clusters.
- defines noise and is thus robust to outliers.

CONs

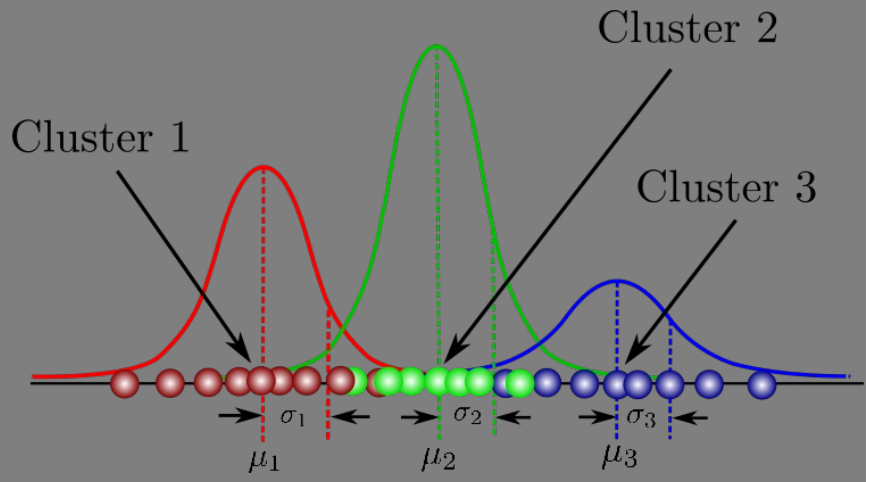
- Sensitive to hyperparameters Rcut and MinPts: cannot cluster well when large differences in densities
- Not deterministic.
- Border points that are reachable from more than one cluster can be part of either cluster.



Density-Based Clustering: DBSCAN

Parametric Unsupervised Clustering

Parametric Unsupervised Learning assumes data comes from a population that follows a probability distribution based on a fixed set of parameters. In practice, each cluster can be mathematically represented by a parametric distribution, like a Gaussian.



The entire data set is modelled by a mixture of these distributions. A mixture model with high likelihood tends to have the following traits:

- distributions in clusters have high "peaks" (data in one cluster are tight); it is like a density-based method in this regard
- mixture model "covers" data well (dominant patterns in the data are captured by component distributions).

Algorithmically it involves construction of Gaussian Mixture Models and using Expectation-Maximization to predict the best clusters.

Gaussian Mixture Models

Gaussian Mixture Models (GMM) are generative models that find the parameters of a set of Gaussians that maximize the likelihood of observing the data.

$$\mathcal{N}(x|m,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} exp\left(-\frac{(x-m)^2}{2\sigma^2}\right)$$

We regard each point as being generated by a Gaussian mixture in which we can compute the probability that it represents x.

$$p(x) = \sum_{i}^{K} \pi_i \mathcal{N}(x|m_i,\sigma_i^2)$$

I.e. each data point x is a linear combination of K Gaussians, and π_i represents their weights, which satisfies the constraint

$$\sum_{i}^{K} \pi_i = 1$$

If all points $X = \{x_1, \dots x_N\}$ are mixtures of K Gaussians then

$$p(X) = \prod_{\mu}^{N} p\left(x^{(\mu)}\right) = \prod_{\mu}^{N} \sum_{i}^{K} \pi_i \mathcal{N}\left(x^{(\mu)}|m_i,\sigma_i^2\right)$$

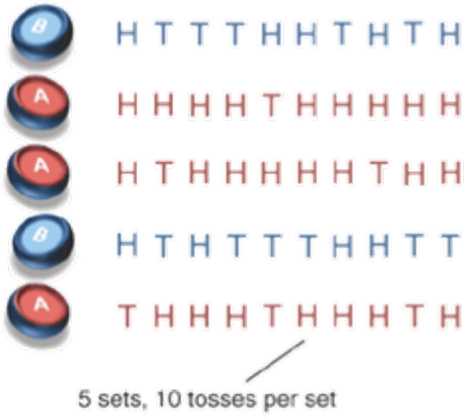
Gaussian Mixture Clustering

Goal: Find $\pi_1, \ldots, \pi_K$ and $m_1, \sigma_1^2, \ldots, m_K, \sigma_K^2$ such that $p(X)$ or $\log p(X)$ is maximized

$$logLikelihood = \sum_{\mu}^{N} log \sum_{i}^{K} \pi_i \mathcal{N}\left(x^{(\mu)} \middle| m_i, \sigma_i^2\right)$$

We have three different parameters that we need to optimize for each Gaussian: π_i, m_i, σ_i^2. How do we learn the parameters? We have seen Maximum Likelihood Estimation before, determining the parameters through optimization that results in the best fit to the data. MLE assumes that the dataset is complete, or fully observed.

Given a pair of coins A and B of unknown biases, θ_A and θ_B, estimate $\theta = (\theta_A, \theta_B)$ by repeating the following procedure five times: randomly choose one of the two coins (with equal probability), and perform ten independent coin tosses with the selected coin.



Coin A	Coin B
	5 H, 5 T
9 H, 1 T	
8 H, 2 T	
	4 H, 6 T
7 H, 3 T	
24 H, 6 T	9 H, 11 T

$$\hat{\theta}_A = \frac{24}{24+6} = 0.80$$

$$\hat{\theta}_B = \frac{9}{9+11} = 0.45$$

Latent Variables

Given a pair of coins A and B of unknown biases, θ_A and θ_B, estimate $\theta = (\theta_A, \theta_B)$ by repeating the following procedure five times: randomly choose one of the two coins (with equal probability), and perform ten independent coin tosses with the selected coin.

But some datasets have only some of relevant variables that are observed, and some are not, and thus they remain hidden. These are latent variables. Expectation-maximization provides an iterative solution to maximum likelihood estimation with latent variables.

Computing proportions of heads for each coin is no longer possible, because we don't know the coin used for each set of tosses.

The latent variables in this case are which coin (with different biases) gave the outcome seen?

Maximum Likelihood Estimation is now much more difficult only knowing the number of Heads and Tail outcomes

Expectation-Maximization

However, if we had some way of completing the data (guessing (in)correctly which coin was used in each of the five sets), then we could reduce parameter estimation for this problem with incomplete data to maximum likelihood estimation with complete data.

Very common algorithm (and most citations ever for a mathematical article)!

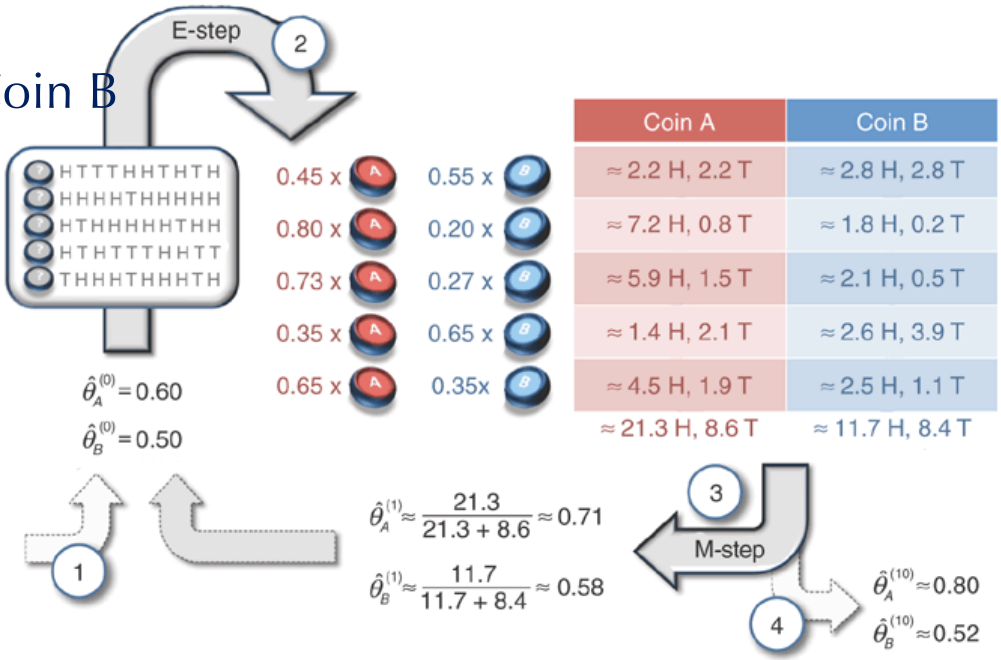
Expectation-maximization estimation with latent variables.

(1) Initialize

(2) Estimate probability θ_A, θ_B of Coin A and Coin B (use binomial probability)

(3) Do Maximum likelihood optimization to find θ_A, θ_B

(4) Converge?
No→return to (2)
Yes→Exit



Gaussian Mixture Clustering

(1) Initialize each point to be a random mixture of K Gaussians, i.e. assign them π_i, m_i, σ_i^2 and evaluate initial value of the loglikelihood

$$logLikelihood = \sum_{\mu}^{N} log \sum_{i}^{K} \pi_i \mathcal{N}\left(x^{(\mu)} \middle| m_i, \sigma_i^2\right)$$

(2) Expectation step: compute the probability that each data point μ was generated by each i Gaussian. We're not assigning each point to a Gaussian, we're determining probability θ_i^{μ} of Gaussian *i* generating a particular point.

$$\theta_i^{\mu} = \frac{\pi_i \mathcal{N}\left(x^{(\mu)} \middle| m_i, \sigma_i^2\right)}{\sum_j^K \pi_j \mathcal{N}\left(x^{(\mu)} \middle| m_j, \sigma_j^2\right)}$$

Note that normalized like a probability

(3) Maximization step. Update π_i, m_i, σ_i^2 for each Gaussian i. To update a mixture weight π_i, we sum up probability θ_i^{μ} that each point μ was generated by Gaussian i and divide by N points (hence are normalized)

$$\pi_i = \frac{1}{N} \sum_{\mu}^{N} \theta_i^{\mu}$$

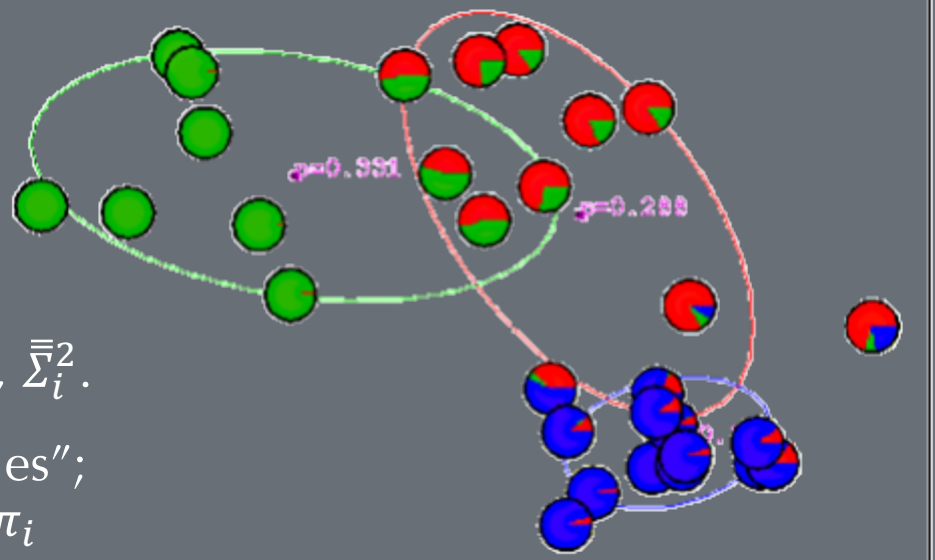
Gaussian Mixture Clustering

(4) We compute mean m_i and (co-)variance σ_i^2 of all points weighted by probability of that point being generated by Gaussian i.

$$m_i = \frac{\sum_\mu^N \theta_i^\mu x^{(\mu)}}{\sum_\mu^N \theta_i^\mu} \qquad \sigma_i^2 = \frac{\sum_\mu^N \theta_i^\mu \left(x^{(\mu)} - m_i\right)\left(x^{(\mu)} - m_i\right)^T}{\sum_\mu^N \theta_i^\mu}$$

(5) Evaluate $logLikelihood$ with new parameters, repeat until converged

$$logLikelihood = \sum_\mu^N log \sum_i^K \pi_i \mathcal{N}\left(x^{(\mu)} \middle| m_i, \sigma_i^2\right)$$



Each colored circle is a $x^{(\mu)}$

- large ellipses represent 3 Gaussian cluster $\vec{m}_i$, $\vec{\vec{\Sigma}}_i^2$.
- Pie chart on $x^{(\mu)}$ correspond to "responsibilities"; i.e. estimates of their Gaussian contributions π_i

Gaussian Mixture Clustering

(4) We compute mean m_i and (co-)variance σ_i^2 of all points weighted by probability of that point being generated by Gaussian i.

$$m_i = \frac{\sum_\mu^N \theta_i^\mu x^{(\mu)}}{\sum_\mu^N \theta_i^\mu} \qquad \sigma_i^2 = \frac{\sum_\mu^N \theta_i^\mu \left(x^{(\mu)} - m_i\right)\left(x^{(\mu)} - m_i\right)^T}{\sum_\mu^N \theta_i^\mu}$$

(5) Evaluate $logLikelihood$ with new parameters, repeat until converged

$$logLikelihood = \sum_\mu^N log \sum_i^K \pi_i \mathcal{N}\left(x^{(\mu)} \middle| m_i, \sigma_i^2\right)$$

Having illustrated the GMM update for isotropic Gaussians, remember we can do the same for anisotropic Gaussians of many dimensions! I.e.

$$\vec{\vec{\Sigma}}_i^2 = \frac{\sum_\mu^N \theta_i^\mu \left(x^{(\mu)} - \vec{m}_i\right)\left(x^{(\mu)} - \vec{m}_i\right)^T}{\sum_\mu^N \theta_i^\mu}$$

This takes most advantage of GMM!

Gaussian Mixture Clustering

PROs

When latent variables are present

Lots of data “looks” Gaussian

Any distribution of interest is reasonably represented as a mixture of Gaussians.

Maximum Likelihood estimation of Gaussians is easy.

Can handle clusters that form elliptical shapes through use of variance

Allows for soft classification by calculating probabilities of data points belonging to clusters.

CONs

Have to guess number of Gaussians

More complicated and thus takes longer.

Very sensitive to initialization