DATA11002 Introduction to Machine Learning

Kai Puolamäki

25 November 2020

Announcements

- Recap lecture on **2 December**
  - I will ask for requests for topics on Friday lecture
- Machine Learning Guest Lectures on **4 December**
  - Andreas Henelius (OP Financial Group): *Data Science in Finance - Machine learning for overdue invoice prediction*
  - Antti Ukkonen (Speechly): *From voice to meaning: Machine learning for spoken language understanding*
  - Discussion

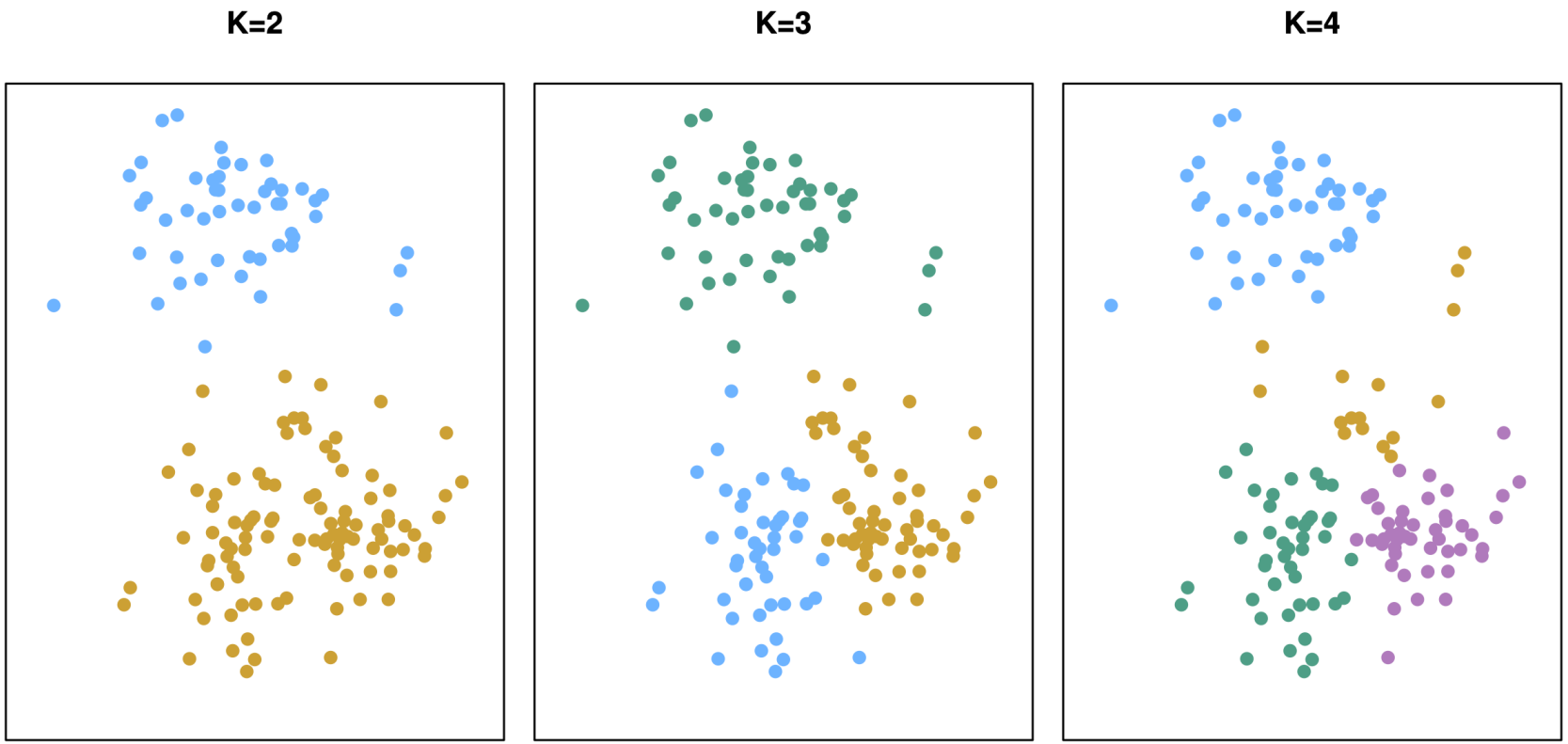
Unsupervised learning

Unsupervised learning

- Unsupervised learning:
  - clustering *(today)*
  - dimensionality reduction *(Friday)*
  - visualization
- No response variable Y, only the variable X
- Several approaches, we will cover:
  - k-means
  - hierarchical clustering

Basic idea (flat clustering)

- Find a clustering (partition) of data points points such that points in the same cluster are "similar" or "nearby" and points in the different clusters are "dissimilar" or "distant".
- Typically, the number of clusters k is selected by user.



James et al., Fig. 10.5.

Basic idea (flat clustering)

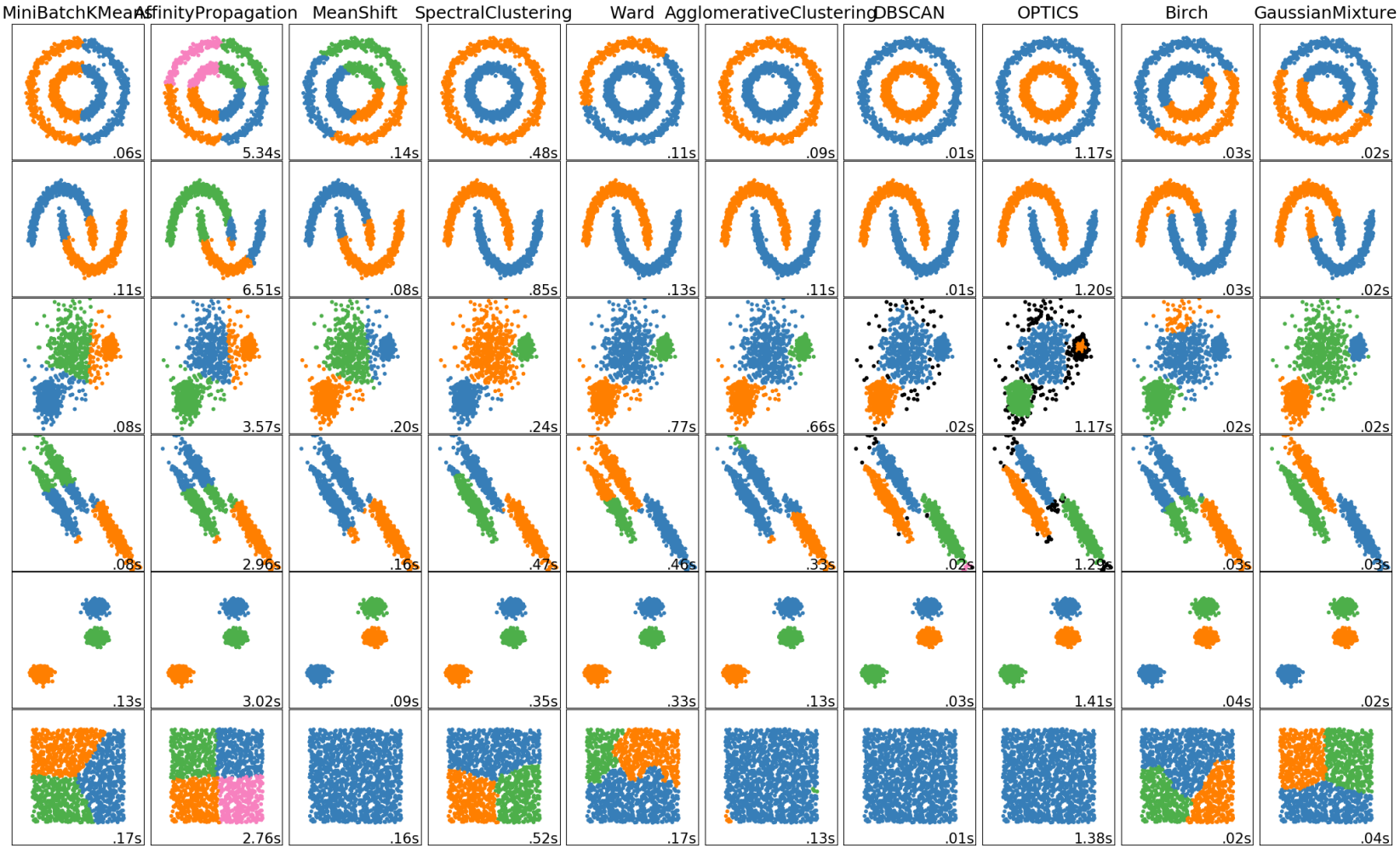


Figure from https://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_comparison.html

Partition or clustering

- Given n data points in $[n] = \{1, \ldots, n\}$ a k-**partition** or k-**clustering** of data points is given by sets $(D_1, \ldots, D_k)$ that satisfy the following:
  - $D_i \subseteq [n]$
  - $D_1 \cup \ldots \cup D_k = [n]$
  - $D_i \cap D_j = \emptyset$ if $i \neq j$

Basic idea (flat clustering)

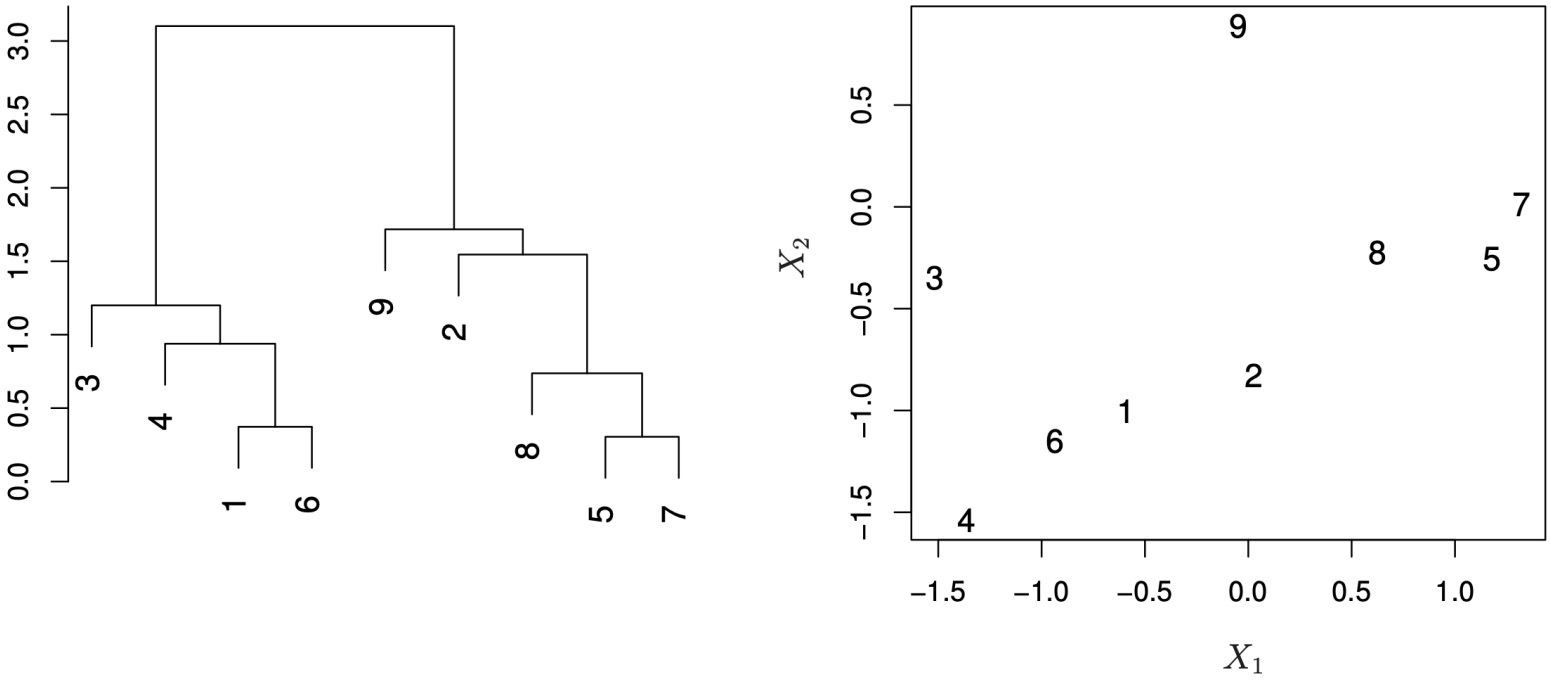
- Clustering can be based on distance
  - . . . but usually distances are (assumed to be) metric, i.e., for any three points a, b, c: $d(a, c) \leq d(a, b) + d(b, c)$
- A common strategy is to represent clusters as k "prototypes" $\mu_1, \ldots, \mu_k$ and assign each point to the closest prototype
  - prototypes are descriptive of clusters (interpretation!), can be used to assign new points to clusters
- Probabilistic topic models *(not covered today)*
  - probabilistic, cluster membership can be probabilistic or "soft"
  - See Blei (2012) Probabilistic topic models. ACM Communications.

Detour: non-negative matrix factorization

- A very related schema is *non-negative matrix factorization*
- Idea: given non-negative design (data) matrix $X \in \mathbb{R}^{n \times p}$ of n p-dimensional observations and an integer k, find non-negative matrices $U \in \mathbb{R}^{n \times k}$ and $V \in \mathbb{R}^{k \times p}$ such that $X \approx UV$.
- Very similar to clustering: leads to k "centroids" (rows of matrix V) which which be interpreted.
- Uses, e.g.: recommender systems and mass spectrometer analysis
- Alternative approaches: clustering, topic models, dimensionality reduction
- See Wang et al. (2013) Nonnegative Matrix Factorization: A Comprehensive Review. IEEE Transactions on Knowledge and Data Engineering.

Basic idea (hierarchical clustering)

- Data points are arranged in a tree
- Horizontal cut of the tree corresponds to a flat clustering



James et al., Fig. 10.10.

Motivation for clustering

- Often number one data anysis algorithm in real world (cluster observations, customers etc.)
- In biology clustering of gene expressions
- Clustering observational data from Hyytiälä
- Trying to figure out a good classification schema (do the clustering and hope that the clusters correspond to the classes)

Other uses (as preprocessing etc. step)

- Reducing the number of data points - e.g., reduce complexity by replacing data points with respective cluster centroids
- Quantization (lossy compression) - save disk space by representing each point by a "close enough" prototype
- Discretization of continuous variables
  - e.g., instead of having combinations of temperature etc., replace them with "typical" daily weathers.

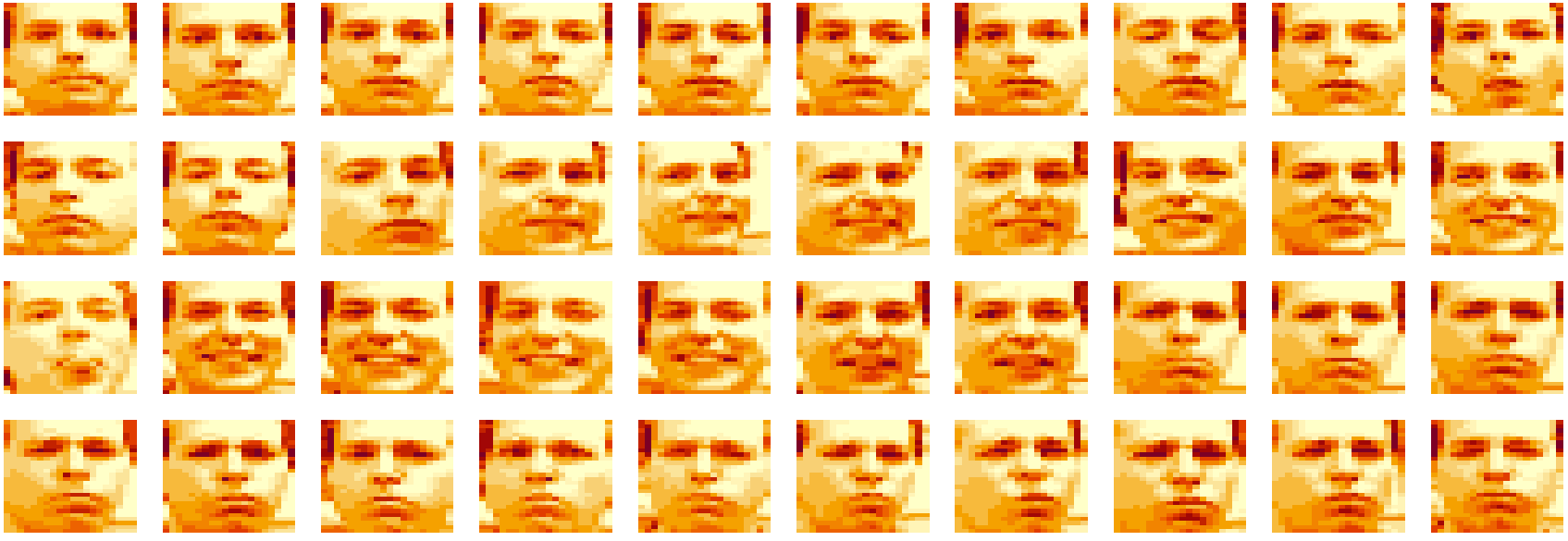
Quantization - image compression

- Image is the set of pixels, each taking 24 bits (3x8 bit numbers in RGB space).
- K-means clustering is applied to the data set of pixels of an image.
- The compressed representation is then the prototype vectors, and cluster index for each pixel.
- k=8 prototypes takes 3 bits each, compression ratio $3/24 = 1/8$!

From left to right: original, 2 prototypes, 3 prototypes, 8 prototypes

Quantization - clustering Frey faces

40 (random but ordered) Frey faces from a short video:



K-means cluster prototype vectors ($k = 5$):

K-means clustering

- Maybe the most popular way to do clustering
- **Assumptions:** data points x are in $\mathbb{R}^p$ and distance measure is the Euclidean distances.
- Given an integer k, find set of centroids $\mu_i \in \mathbb{R}^p$ such that the following loss is minimised

$$\mathcal{L}_{kmeans} = \sum_{j=1}^{k} \sum_{i \in D_j} \|x_i - \mu_j\|_2^2 = \sum_{i=1}^{n} \|x_i - \mu_{j(i)}\|_2^2.$$

- The optimisation problem is NP-hard and therefore usually impossible to solve exactly in practice
  - We use heuristic algorithms (like with the subset selection we used forward selection) that may end up in local optimum

Soft clustering

- In *soft* clustering we would give cluster memberships a weight $r_{ij} \in [0, 1]$ such that $\sum_{j=1}^{k} r_{ij} = 1$ for all i and minimise

$$\mathcal{L}_{soft} = \sum_{i=1}^{n} r_{ij} \|x_i - \mu_j\|_2^2.$$

- *Hard* clustering discussed here is a special case where $r_{ij} \in \{0, 1\}$.
- Notice that optimum assignents for this loss (without any additional terms) are always hard!

Soft clustering (extra material)

- Probabilitic model (one of many possibilities) for the likelihood of the data:

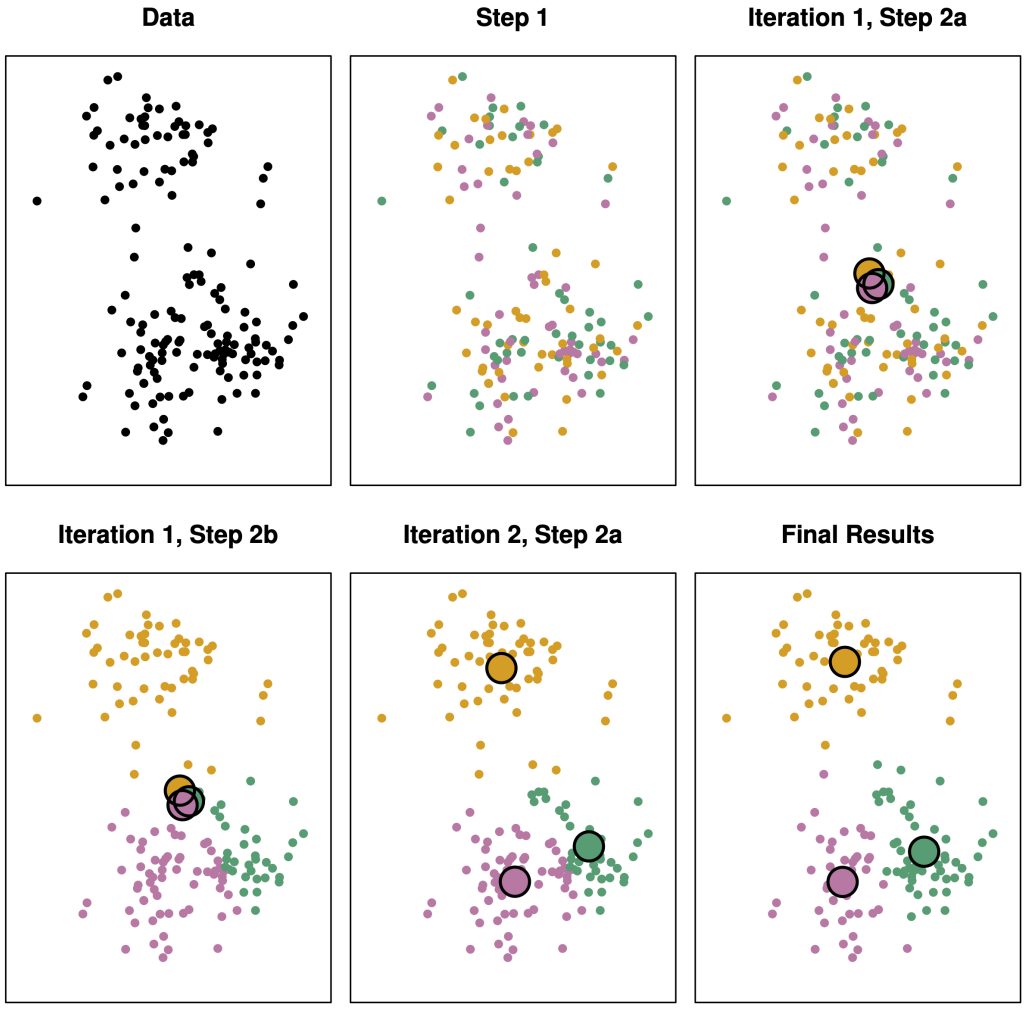
$$P(x \mid \mu_1, \ldots, \mu_k, \pi) = \prod_{i=1}^{n} p(x_i \mid \mu_{z_i}) p(z_i \mid \pi).$$

- $z_i \in [k]$ is the cluster assignment of data point i, $p(x_i \mid \mu_{z_i})$ is the (cluster-specific) unit-variance normal distribution with mean vector $\mu_{z_i} \in \mathbb{R}^p$, and $p(z_i \mid \pi) \in [0, 1]$ is the probability of cluster z_i (multinomial distribution).
- You could, e.g.,find a ML solutions using the expectation maximization (EM) algorithm, see, e.g., Hastie et al., Sec. 8.5.
- Would result to soft clustering (there would typically be uncertainty in the class memberships z_i).

Lloyd's (k-means) algorithm

- Lloyd's algorithm is the commonly used heuristic algorithm to solve the kmeans optimization problem
  - there are also other algorithms to solve the optimization problem
- First, assign data points to random clusters
  - Alternatively, initialize with random cluster centroids
- Alternate between the following steps until convergence:
  - Set each cluster centroid to average of the points in the cluster
  - Assign each data point to the closest centroid

Lloyd's (k-means) algorithm



James et al., Fig. 10.6.

Lloyd's algorithm: convergence

- Lloyd's algorithm always converges (stops) after finite number of steps

$$\mathcal{L}_{kmeans} = \sum_{j=1}^{k} \sum_{i \in D_j} \|x_i - \mu_j\|_2^2 = \sum_{i=1}^{n} \|x_i - \mu_{j(i)}\|_2^2.$$

- We can prove this by showing that the **loss cannot increase** at either step of the algorithm:
  - When we assign the point i to the closest centroid which minimises $\|x_i - \mu_{j(i)}\|_2^2$.
  - When we choose μ_j to be mean of the points this minimises $\|x_i - \mu_j\|_2^2$.
    - Recall: choosing the mean vector minimizes the sum of squared errors!
  - When the loss no longer changes (after two steps above) the algorithm stops.
- Hence, the loss decreases after every two steps and because there are **finite number of states** (possible assignments of points to clusters) the **algorithm must stop after finite time**.

Lloyd's algorithm: space and time complexity

- Space requirements are modest. In addition to data you need to store...
  - cluster indices for each data point
  - cluster centroids
- Running time is linear $O(Mnkp)$, where M is number of iterations, n is the number of data points, k is the number of clusters, and p is the dimensionality of data.
- But what is the number of iterations M?
  - naive upper bound: there are k^n possible combinations of cluster assignments.
  - However, in practice the algorithm usually converges quite fast.
    - ...but not always, the worst-case bounds for M are larger than are practically computable (in the worst case run time is $O(n^{k+2/p})$, Arthur et al. (2006) How slow is the k-means method? In Proc SGC '06)

Lloyd's algorithm: the effect of initialisation

- Recall: we are not guaranteed to find the global optimum. Initialization matters!
- Results of different runs on the same data:

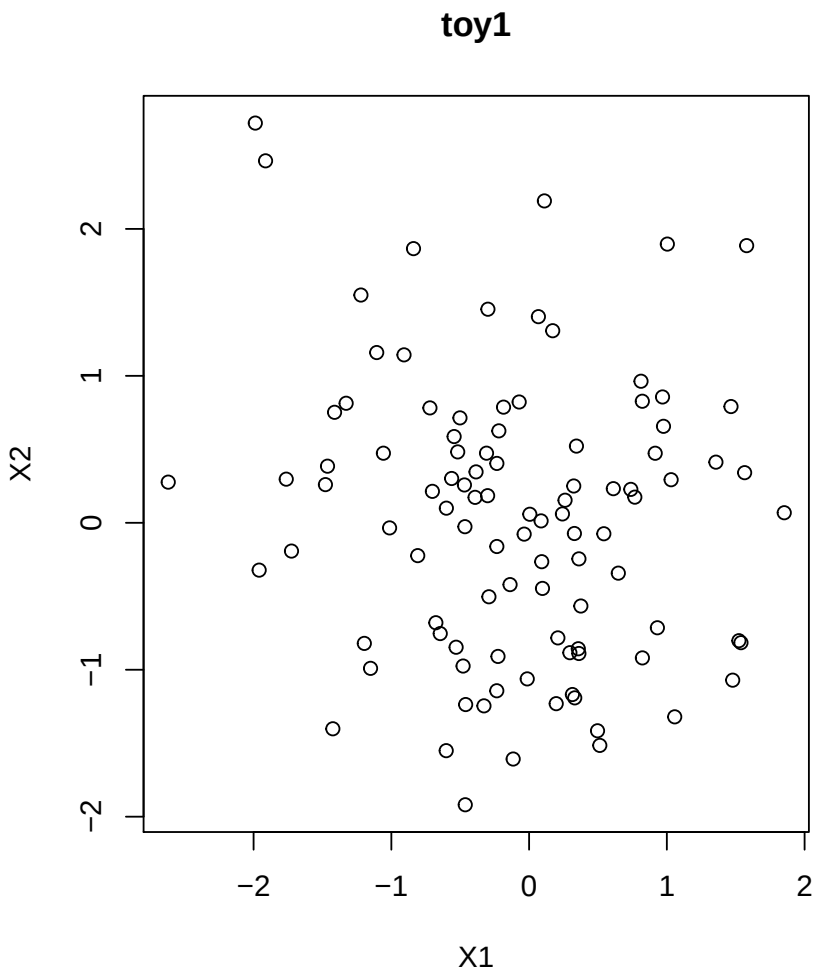
James et al., Fig. 10.7.

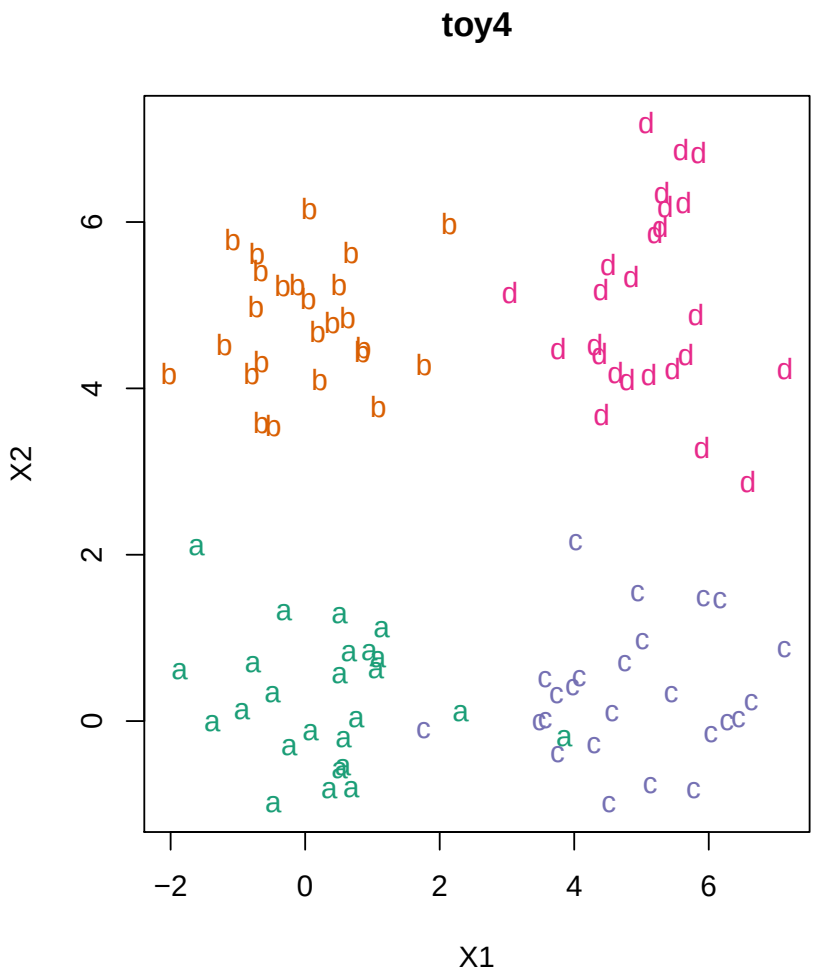
Lloyd's algorithm: the effect of initialisation

- In the worst case the result can be quite bad (no approximation ratio)
  - Consider $n = 4$ toy data and $k = 2$ clusters: $x_1 = (-1, -\epsilon)$, $x_2 = (-1, \epsilon)$, $x_3 = (1, -\epsilon)$, $x_4 = (1, \epsilon)$.
  - Optimal solution: $\mu_1 = (-1, 0)$, $\mu_2 = (1, 0)$, loss $\mathcal{L}^{OPT}_{kmeans} = 4\epsilon^2$.
  - Possible solution: $\mu_1 = (0, -\epsilon)$, $\mu_2 = (0, \epsilon)$, loss $\mathcal{L}^{ALG}_{kmeans} = 4$.
  - $\mathcal{L}^{ALG}_{kmeans} / \mathcal{L}^{OPT}_{kmeans} \to \infty$ as $\epsilon \to 0$!
- **Solution 1:** run the algorithm several times and pick a solution with smallest loss
- **Solution 2:** initialise centroids properly
  - Cool initialisation (default in sklearn), kmeans++, Arthur et al. (2007).
  - Provides (expected) approximation ratio, i.e., upper bound for $\mathcal{L}^{ALG}_{kmeans} / \mathcal{L}^{OPT}_{kmeans}$.
  - Still run some times and pick the solutions with smallest loss.

How to select number of clusters

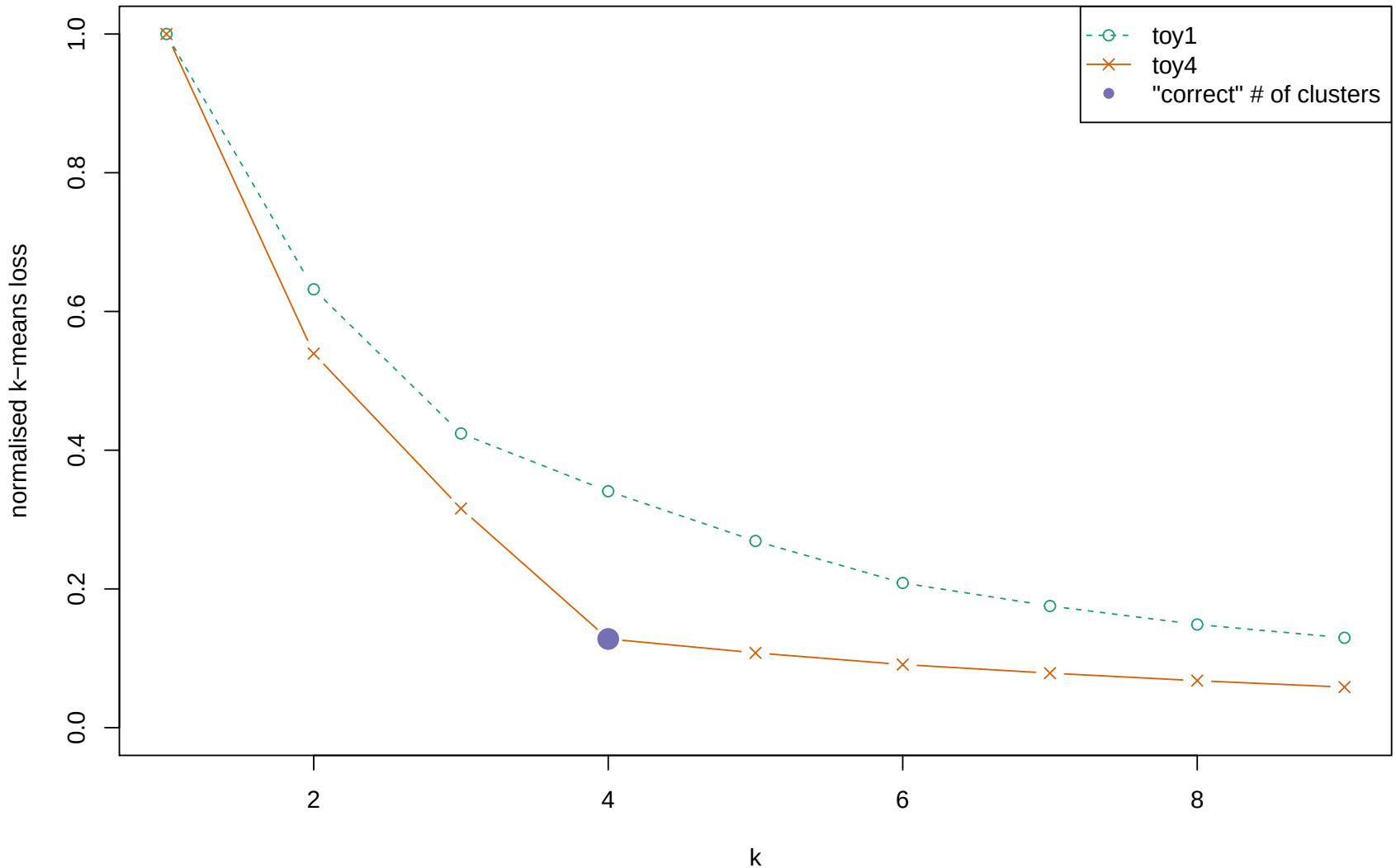
- There is no clear best answer
- There are several approaches. The most common is to look for a "kink" (corner) in the k vs. loss curve.





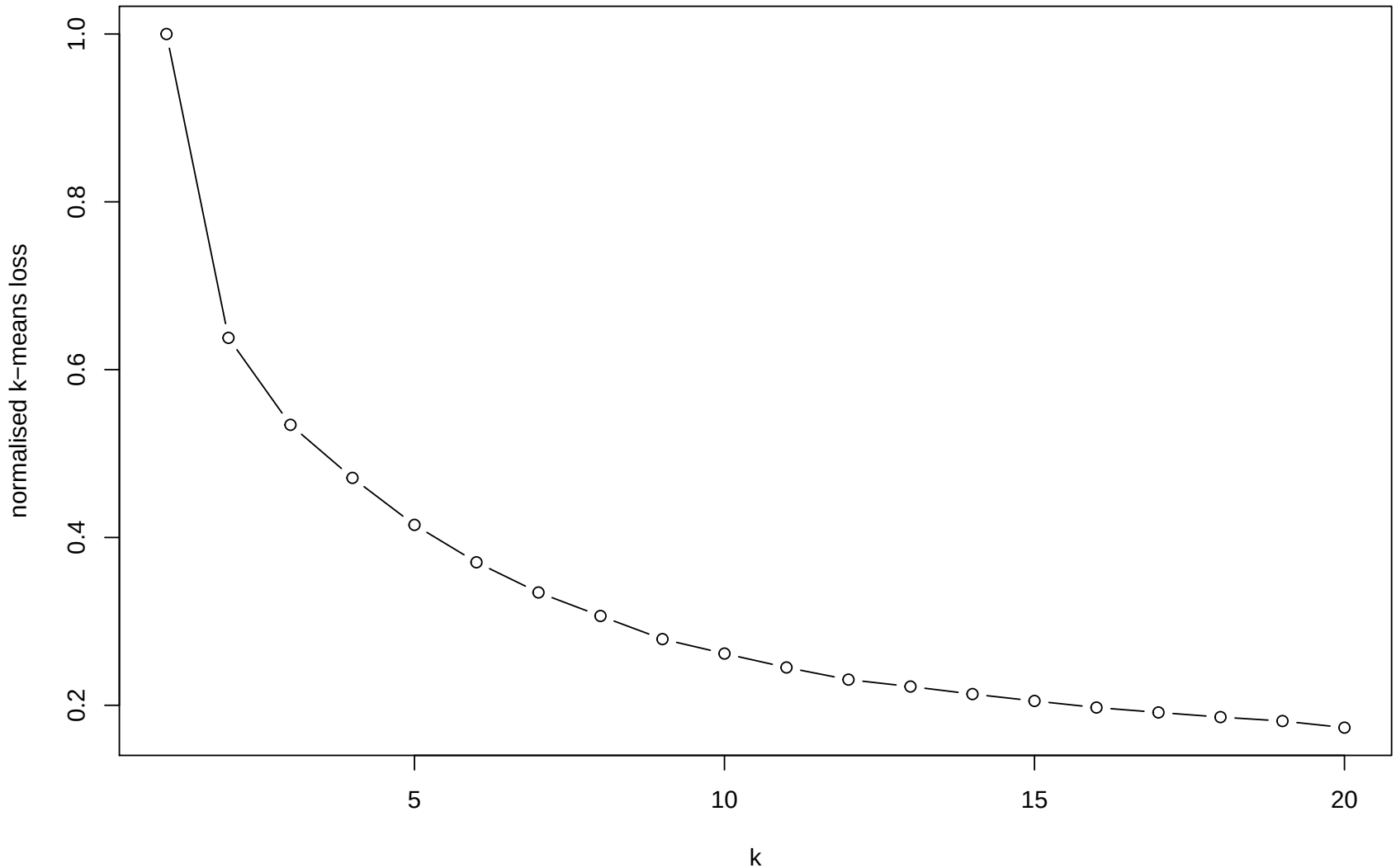
How to select number of clusters: toy data

```
loss1 <- sapply(1:9,function(k)
  kmeans(toy1[,1:2],k,iter.max=100,nstart=100,algorithm="Lloyd")$tot.withinss)
loss4 <- sapply(1:9,function(k)
  kmeans(toy4[,1:2],k,iter.max=100,nstart=100,algorithm="Lloyd")$tot.withinss)
```



How to select number of clusters: Boston data

```
library(MASS); set.seed(42)
lossb <- sapply(1:20,function(k) kmeans(scale(Boston[,1:13]),k,iter.max=100,nstart=100,
                                        algorithm="Lloyd")$tot.withinss)
plot(lossb/lossb[1],type="b",xlab="k",ylab="normalised k-means loss")
```



Matching clusters with class variables or other clusterings

- Often you want to match clusters, e.g., to known class indices
- *Hungarian algorithm* can find a permutation of cluster indices so that the sum on the diagonal of the contingency table of class (a-d) vs. cluster indices (1-4) is maximised
  - implemented, e.g., by `solve_LSAP` in R library `clue` or `linear_sum_assignment` in Python library `scipy.optimize`

```
library(clue); set.seed(42)
cl <- kmeans(toy4[,1:2],4,iter.max=100,nstart=100,algorithm="Lloyd")
tt <- table(toy4$c,cl$cluster)
i <- solve_LSAP(tt,maximum=TRUE)
print(i)
```

```
## Optimal assignment:
## 1 => 3, 2 => 1, 3 => 2, 4 => 4
```

```
knitr::kable(list(tt,tt[,i]))
```

	1	2	3	4
a	0	1	24	0
b	25	0	0	0
c	0	24	1	0
d	0	0	0	25

	3	1	2	4
a	24	0	1	0
b	0	25	0	0
c	1	0	24	0
d	0	0	0	25

Left: original contingency table, right: contingency table with cluster indices permuted

How to interpret clusters?

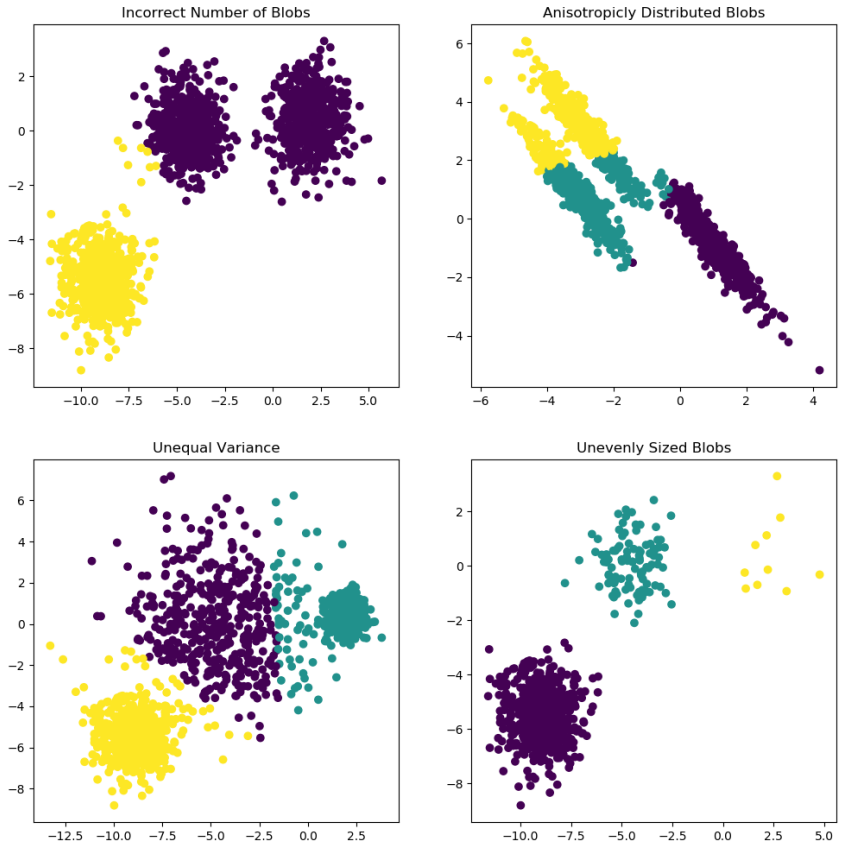
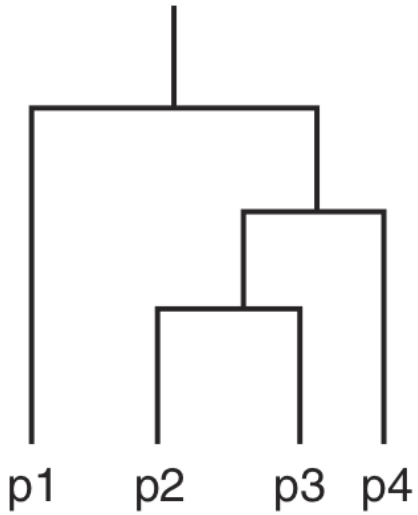
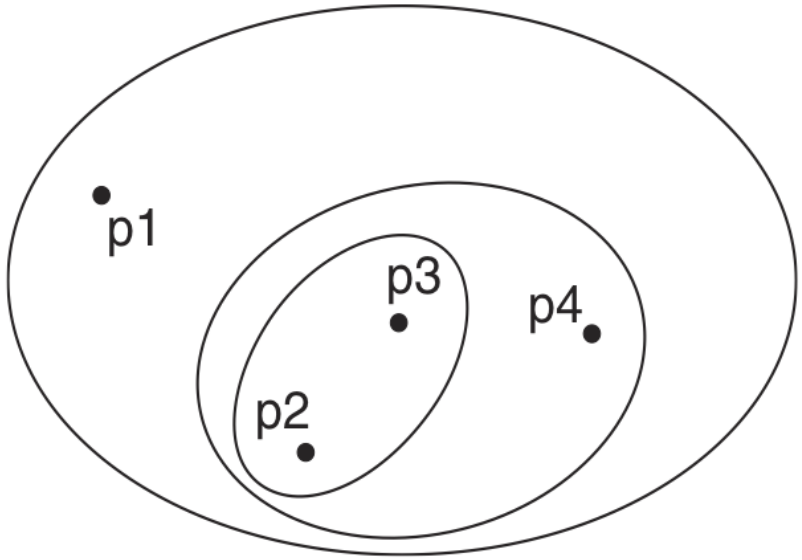


Figure from https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_assumptions.html

Hierarchical clustering

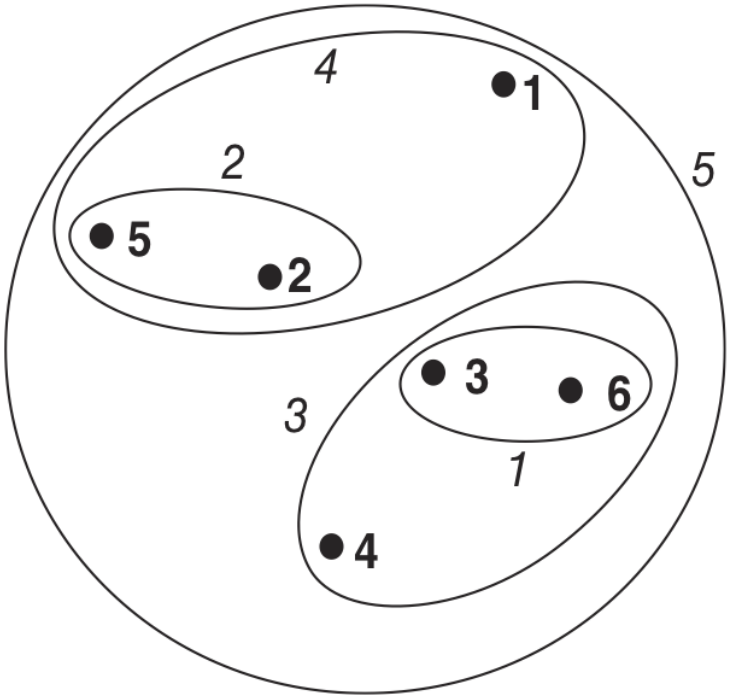
- Dendograms (trees)
- *Nested* cluster structure
- Binary tree with datapoints as leaves
- Cutting tree at any level produces a flat clustering
- Adding new points to "clusters" afterwards is difficult (no cluster centroids etc.)

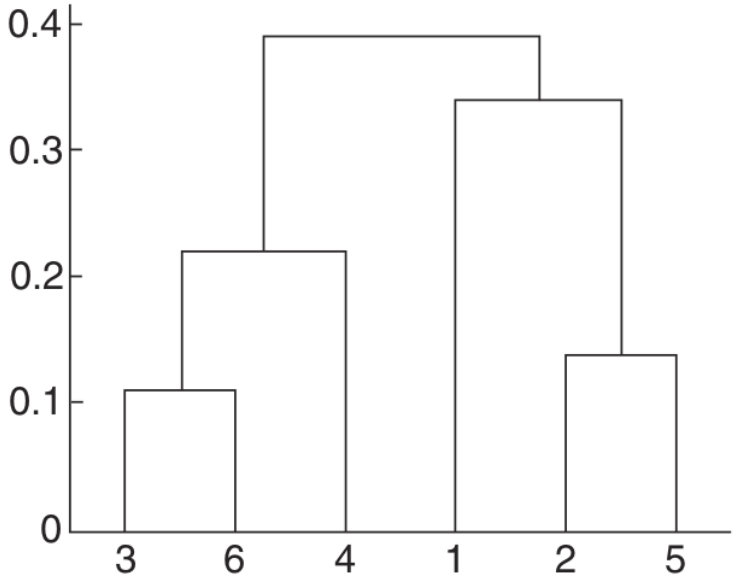




Hierarchical clustering

- Height of the horizontal connectors indicates dissimilarity between the combined clusters (details later)





Hierarchical clustering

- Algorithmic clustering - usually we have no explicit cost function to optimize
- General approaches to find the dendrogram:
  - **divisive approach**
    - start with all points in one big cluster
    - split the cluster recursively into two (e.g., by k-means) until all data points are in singleton clusters
  - **agglomerative approach** (much more common, we focus on this here)
    - start with all points in singleton clusters
    - iteratively merge two most similar clusters until all data points are in one big cluster

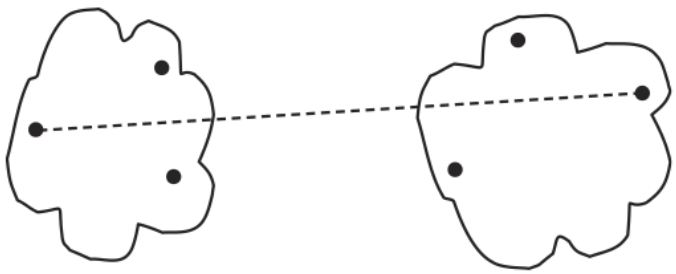
Linkage functions

- We need a *linkage function* that measures (dis)similarity between two clusters
- The 4 most common linkage functions are:
  - **complete** linkage
  - **single** linkage
  - **average** linkage
  - **centroid** linkage (**Ward**)
- complete, single, average linkage require distance function $d(i, j)$ between data points i and j
  - impractical for large datasets ($n \gg 10^3$).

Complete linkage

- *All* points in the cluster should be reasonably close to each other (i.e., no outliers)

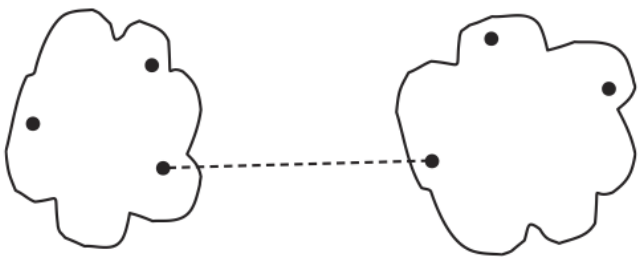
$$L_{complete}(A, B) = \max_{i \in A, j \in B} d(i, j).$$



Single linkage

- *One* pair of the points in the clusters should be close to each others

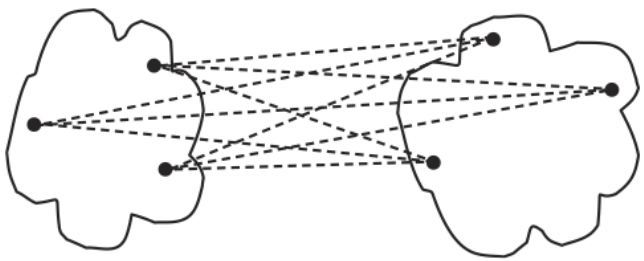
$$L_{single}(A, B) = \min_{i \in A, j \in B} d(i, j).$$



Average linkage

- An intermediate criterion is averaging

$$L_{average}(A, B) = \sum_{i \in A} \sum_{j \in A} d(i, j)/(|A||B|).$$



Centroid linkage

- Requires that we can compute mean vectors of data points, i.e., data is vectors.
- Complete, single, average linkage work with just distance measure, no vectorial data required!

$$L_{centroid}(A, B) = \mathrm{dist}(\mu_A, \mu_B),$$

- . . . where $\mu_A = \sum_{i \in A} x_i / |A|$ and $\mu_B = \sum_{i \in B} x_i / |B|$ are the cluster mean vectors.

Ward

- This is very similar to k-means (and often leads to k-means-looking results in practice!)
- Distance = how much sum of squares will increase if clusters are merged

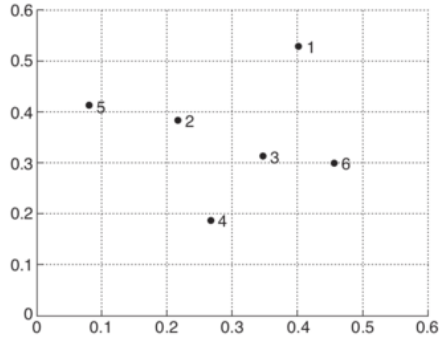
$$
\begin{aligned}
&L_{ward}(A, B) \\
&\qquad = \sum_{i \in A \cup B} |x_i - \mu_{A \cup B}|^2 - \sum_{i \in A} |x_i - \mu_A|^2 - \sum_{i \in B} |x_i - \mu_B|^2 \\
&\qquad = \frac{|A||B|}{|A \cup B|} |\mu_A - \mu_B|^2
\end{aligned}
$$

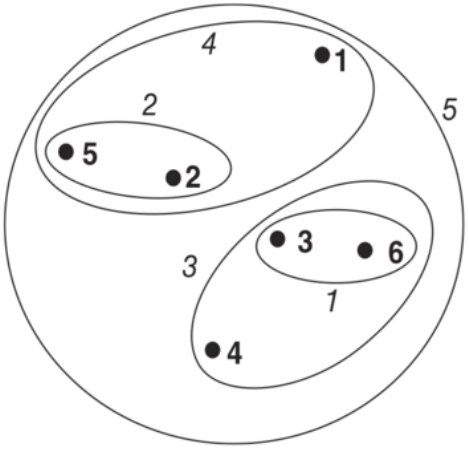
- NB: There is some variation in terminology and implementations, see, e.g., discussion about `Ward.D` vs `Ward.D2` in https://stat.ethz.ch/R-manual/R-devel/library/stats/html/hclust.html

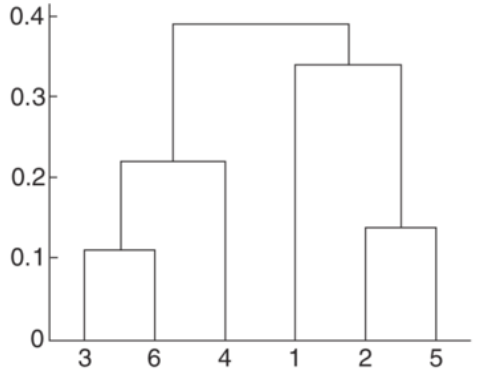
Example: complete linkage

- Heights in the dendrogram correspond to linkage function value when clusters are merged

	p1	p2	p3	p4	p5	p6
p1	0.00	0.24	0.22	0.37	0.34	0.23
p2	0.24	0.00	0.15	0.20	0.14	0.25
p3	0.22	0.15	0.00	0.15	0.28	0.11
p4	0.37	0.20	0.15	0.00	0.29	0.22
p5	0.34	0.14	0.28	0.29	0.00	0.39
p6	0.23	0.25	0.11	0.22	0.39	0.00



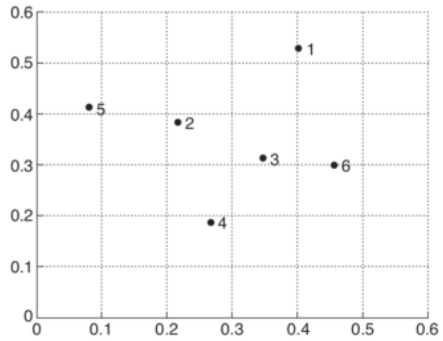


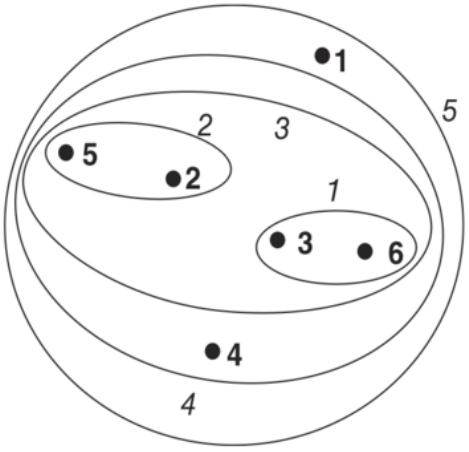


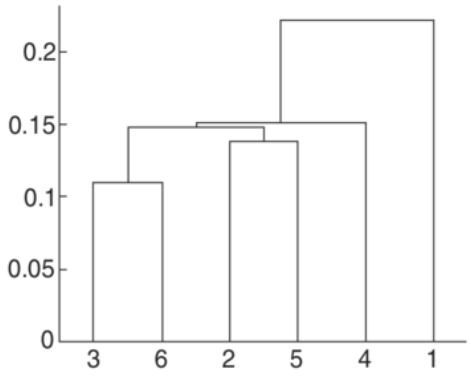
Example: single linkage

- Heights in the dendrogram correspond to linkage function value when clusters are merged

	p1	p2	p3	p4	p5	p6
p1	0.00	0.24	0.22	0.37	0.34	0.23
p2	0.24	0.00	0.15	0.20	0.14	0.25
p3	0.22	0.15	0.00	0.15	0.28	0.11
p4	0.37	0.20	0.15	0.00	0.29	0.22
p5	0.34	0.14	0.28	0.29	0.00	0.39
p6	0.23	0.25	0.11	0.22	0.39	0.00



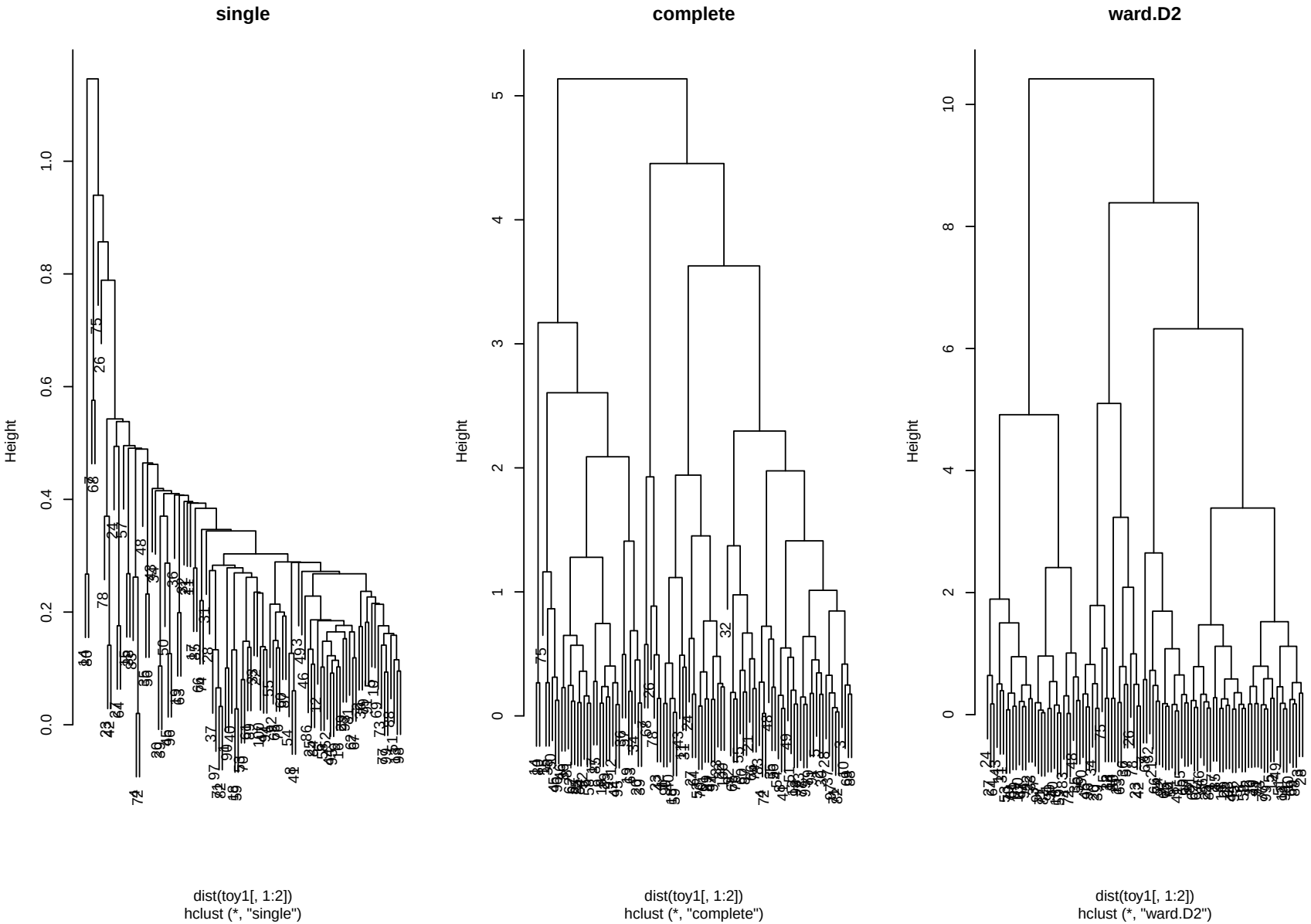




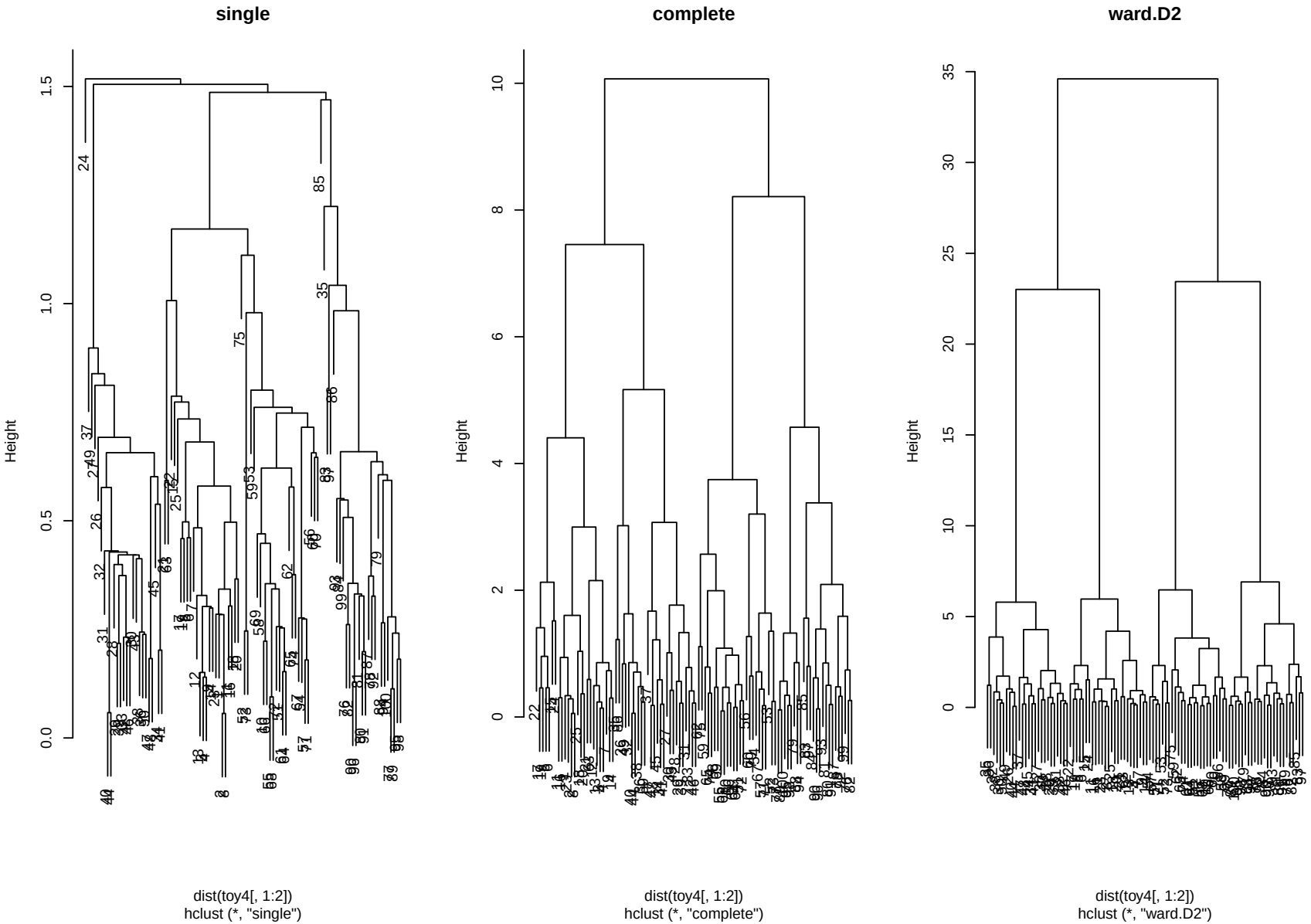
Cluster shapes

- **Complete-linkage** tends to produce even-sized clusters. If there are outliers they will be in their own clusters. Use if you want to cover the data space with even-sized clusters.
- **Single-linkage** can produce arbitrarily shaped clusters (joining quite different objects which have some intermediate links that connect them). Can find and separate continuous (and even intermingled) regions, but is sensitive to noise (even one random point can ruin nice clusters).
- **Ward** (like k-means) tends to produce clusters with a central area. Clusters may contain some outliers (maybe distant).
  - Ward method is often a good alternative for k-means (you get hierarchical structure for free!)

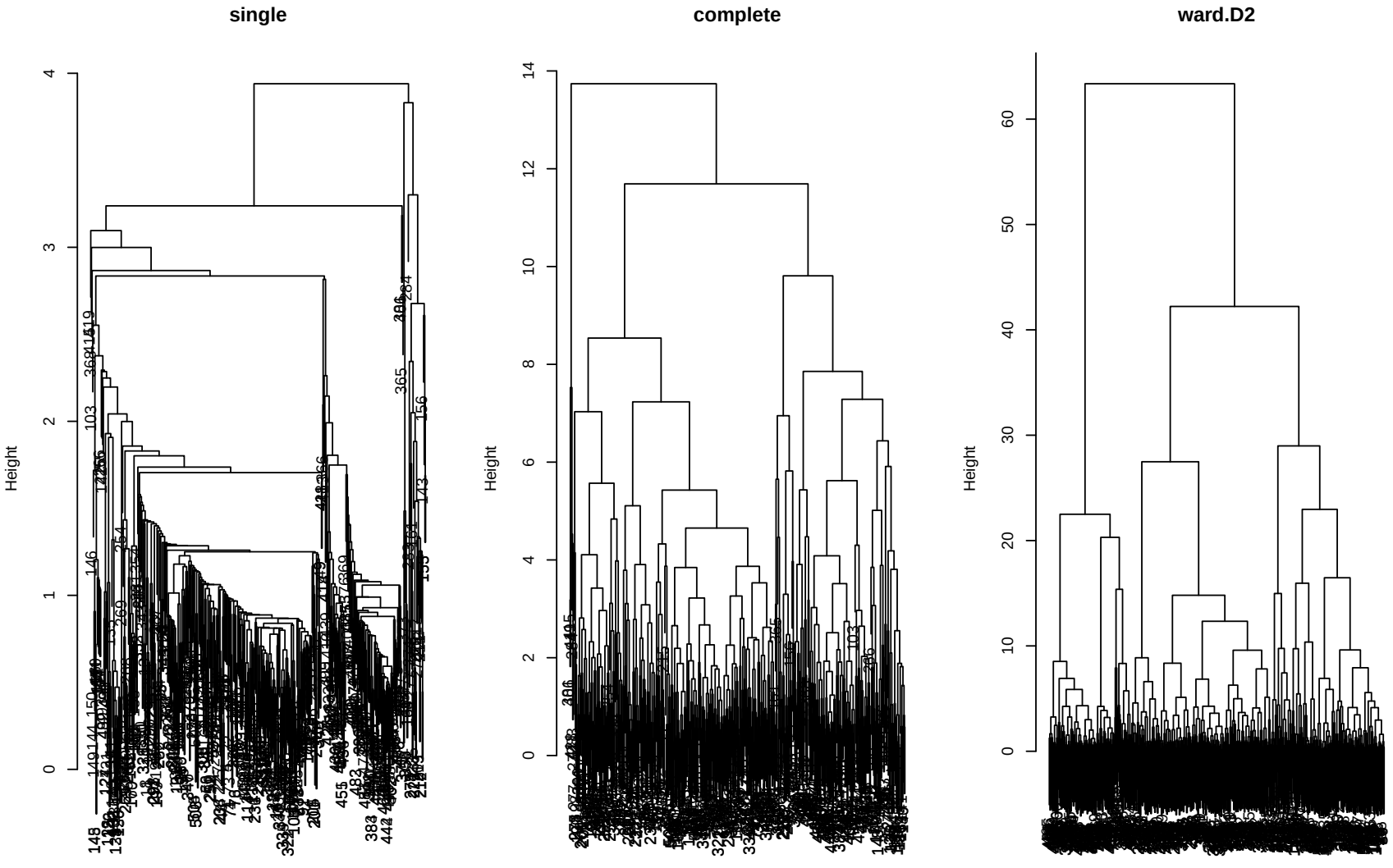
Hierarchical clustering: toy1



Hierarchical clustering: toy4



Hierarchical clustering: Boston



Hiearchical clustering

- No obvious global objective function
- Difficult to add points afterwards
- Computational complexity (if distance matrix is needed)
  - Storing distance matrix takes $O(n^2)$ memory
  - Finding the dendrogram takes $O(n^2) \dots O(n^3)$ time
  - Applicable for relatively small datasets!