Introduction to Machine Learning

Clustering

Bernd Bischl, Christoph Molnar, Daniel Schalk, Fabian Scheipl

Department of Statistics – LMU Munich

Hierarchical Clustering

MOTIVATION FOR CLUSTERING

Consider multivariate data with N observations (e.g. customers) and P features (e.g. characteristics of customers).
Task: divide data into groups (clusters), such that

- the observations in each cluster are as "similar" as possible (homogeneity within each cluster), and
- the clusters are as "far away" as possible from other clusters (heterogeneity between different clusters).

CLUSTERING VS. CLASSIFICATION

- In classification, the groups are known and we try to learn what differentiates these groups (i.e., learn a classification function) to properly classify future data.
- In clustering, we look at data, where groups are unknown and try to find similar groups.

Why do we need clustering?

- Discovery: looking for new insights in the data (e.g. finding groups of customers that buy a similar product).
- Derive a reduced representation of the full data set.

HIERARCHICAL CLUSTERING

Hierarchical clustering is a recursive process that builds a hierarchy of clusters. We distinguish between:

1. Agglomerative (or bottom-up) clustering:
    - Start: Each observations is an *individual cluster*.
    - Repeat: Merge the two closest clusters.
    - Stop when there is only one cluster left.
2. Divisive (or top-down) clustering:
    - Start: All observations are within *one* cluster.
    - Repeat: Divide the cluster that results in two clusters with biggest distance.
    - Stop when each observation is an individual cluster.

HIERARCHICAL CLUSTERING

Let $X_1, \dots, X_N$ be observations with P features (dimensions), where $X_i = (x_{i1}, \dots, x_{iP})^\top$.
A data set is a (N $\times$ P)-matrix of the form:

	feature 1	$\dots$	$\dots$	feature P
X_1	x_{11}	$\dots$	$\dots$	x_{1P}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
X_N	x_{N1}	$\dots$	$\dots$	x_{NP}

HIERARCHICAL CLUSTERING

For hierarchical clustering, we need a definition for

- distances $d(X_i, X_j)$ between two observations X_i and X_j:
    - manhattan distance:

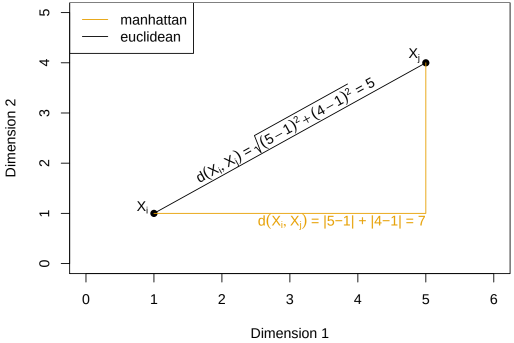
$$d(X_i, X_j) = ||X_i - X_j||_1 = \sum_{k=1}^{P} |x_{ik} - x_{jk}|$$

    - euclidean distance:

$$d(X_i, X_j) = ||X_i - X_j||_2 = \sqrt{\sum_{k=1}^{P} (x_{ik} - x_{jk})^2}$$

- distances between two clusters (called linkage).

DISTANCES BETWEEN OBSERVATIONS



- manhattan: sum up the absolute distances in each dimension. In R: “dist(data, method = "manhattan")”
- euclidean: remember Pythagoras theorem from school? In R: “dist(data, method = "euclidean")”
- gower: can be used for mixed variables (categorical and numeric). In R: “gower_dist()” from the “gower” package
- see “?dist” for other distances.

GOWER DISTANCE I

- The Gower's metric calculates the distance between observations X_i and X_j for each feature separately and based on its data type (i.e., categorical or numeric).
- For a categorical feature X_k, the distance between the *i*-th and the *j*-th observation X_{ik} and X_{jk} is defined by

$$s_{ijk} = \begin{cases} 0 \text{ if } X_{ik} = X_{jk} \\ 1 \text{ if } X_{ik} \neq X_{jk}. \end{cases}$$

- For a numerical feature X_k, the distance between the *i*-th and the *j*-th observation X_{ik} and X_{jk} is defined by

$$s_{ijk} = \frac{|X_{ik} - X_{jk}|}{\max(X_k) - \min(X_k)}, \quad \text{so that } 0 \leq s_{ijk} \leq 1.$$

GOWER DISTANCE II

The Gower's metric S combines all individual distances of each feature by

$$S_{ij} = \frac{\sum_{k=1}^{P} w_k s_{ijk}}{\sum_{k=1}^{P} w_k},$$

where

- P: number of features.
- w_k: weight for feature k (typically $w_k = 1$).
- s_{ijk}: the difference (distance) between X_{ik} and X_{jk}, i.e. the i-th and j-th observation of feature k.

GOWER DISTANCE III - EXAMPLE

Observation	X_1	X_2
1	4	a
2	3	b
3	6	a

$w_k = 1$ for $k \in \{1, 2\}$

$s_{121} = \frac{|4-3|}{6-3} = \frac{1}{3}$

$s_{122} = 1$, since a≠b

$$S_{12} = \frac{\frac{1}{3} + 1}{2} = \frac{2}{3}$$

$s_{131} = \frac{|4-6|}{6-3} = \frac{2}{3}$

$s_{132} = 0$, since a=a

$$S_{13} = \frac{\frac{2}{3} + 0}{2} = \frac{1}{3}$$

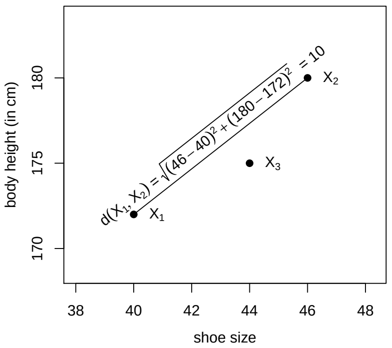
$s_{231} = \frac{|3-6|}{6-3} = 1$

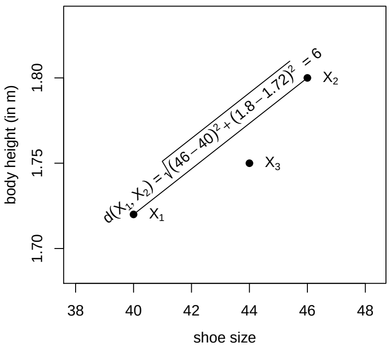
$s_{232} = 1$, since b≠a

$$S_{23} = \frac{1 + 1}{2} = 1$$

DISTANCES BETWEEN OBSERVATIONS

It is often a good idea to *normalize* the data before computing distances, especially when the scale of features is different, e.g.:





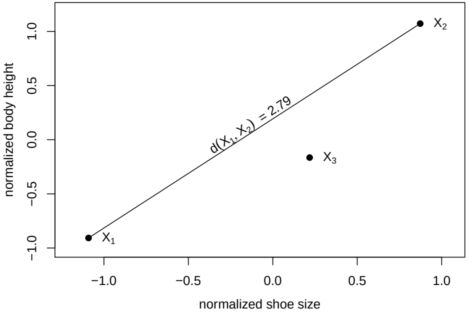
On the right plot, the distance is dominated by ‘shoe size‘.

DISTANCES BETWEEN OBSERVATIONS

The normalized feature $\tilde{X}_{\texttt{height}}$ is computed using $X_{\texttt{height}}$ by <!– Normalization of the `height` feature means: –>

$$\tilde{X}_{\texttt{height}} = \frac{X_{\texttt{height}} - \texttt{mean}(X_{\texttt{height}})}{\texttt{sd}(X_{\texttt{height}})}.$$

Distances based on normalized data are better comparable and robust in terms of linear transformations (e.g. unit conversion).



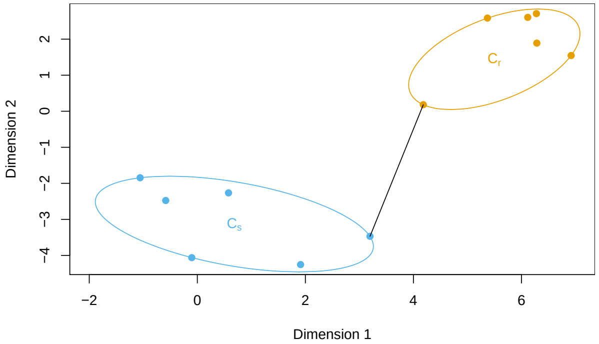
DISTANCES BETWEEN CLUSTERS (LINKAGE)

- Assume that all observations $X_1, \dots, X_N$ belong to $K < N$ different clusters.
- The linkage of two clusters C_r and C_s is a "score" describing their distance.

The most popular and simplest linkages are

- *Single Linkage*
- *Complete Linkage*
- *Average Linkage*
- *Centroid Linkage*

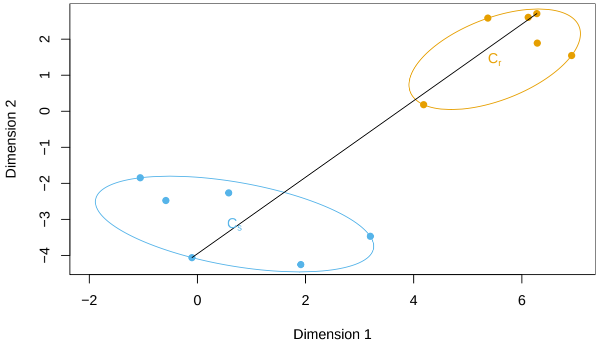
SINGLE LINKAGE



Single linkage defines the distance of the *closest point pairs* from different clusters as the distance between two clusters:

$$d_{\text{single}}(C_r, C_s) = \min_{i \in C_r, j \in C_s} d(X_i, X_j)$$

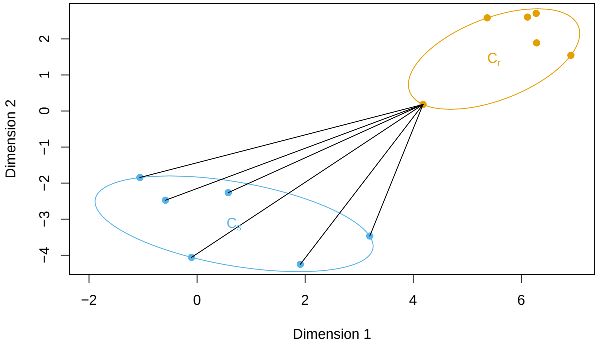
COMPLETE LINKAGE



Complete linkage defines the distance of the *furthest point pairs* of different clusters as the distance between two clusters:

$$d_{\text{complete}}(C_r, C_s) = \max_{i \in C_r, j \in C_s} d(X_i, X_j)$$

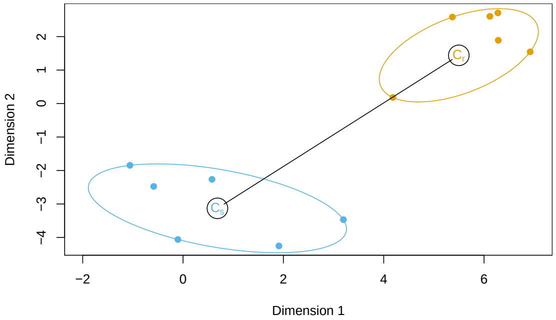
AVERAGE LINKAGE



(Note: Plot only shows distances between all green points and *one* red point)

In average linkage, the distance between two clusters is defined as the average distance across *all* pairs of two different clusters.

CENTROID LINKAGE



Centroid linkage defines the distance between two clusters as the distance between the two cluster centroids. The centroid of a cluster C_s with N_s points is the mean value of each dimension:

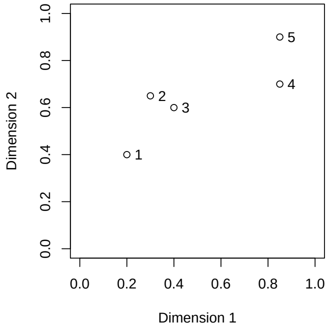
$$\bar{X}_s = \frac{1}{N_s} \sum_{i \in C_s} X_i$$

EXAMPLE: HIERARCHICAL CLUSTERING

Agglomerative hierarchical clustering starts with all points forming their own cluster and iteratively merges them until all points form a single cluster containing all points.

Example:

Step 1: $\{1\}, \{2\}, \{3\}, \{4\}, \{5\}$

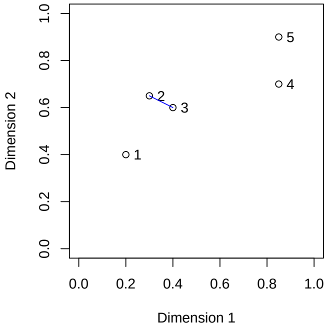


EXAMPLE: HIERARCHICAL CLUSTERING

Agglomerative hierarchical clustering starts with all points forming their own cluster and iteratively merges them until all points form a single cluster containing all points.

Example:

Step 1: $\{1\},\{2\},\{3\},\{4\},\{5\}$
Step 2: $\{1\},\{2,3\},\{4\},\{5\}$

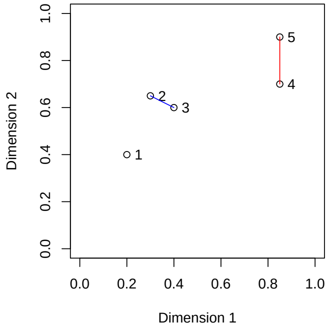


EXAMPLE: HIERARCHICAL CLUSTERING

Agglomerative hierarchical clustering starts with all points forming their own cluster and iteratively merges them until all points form a single cluster containing all points.

Example:

Step 1: $\{1\}, \{2\}, \{3\}, \{4\}, \{5\}$
Step 2: $\{1\}, \{2, 3\}, \{4\}, \{5\}$
Step 3: $\{1\}, \{2, 3\}, \{4, 5\}$

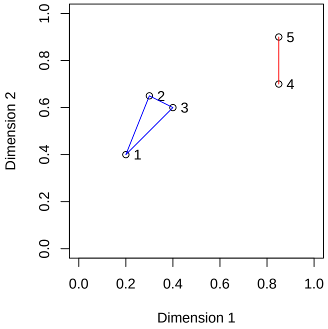


EXAMPLE: HIERARCHICAL CLUSTERING

Agglomerative hierarchical clustering starts with all points forming their own cluster and iteratively merges them until all points form a single cluster containing all points.

Example:

Step 1: $\{1\}, \{2\}, \{3\}, \{4\}, \{5\}$
Step 2: $\{1\}, \{2, 3\}, \{4\}, \{5\}$
Step 3: $\{1\}, \{2, 3\}, \{4, 5\}$
Step 4: $\{1, 2, 3\}, \{4, 5\}$

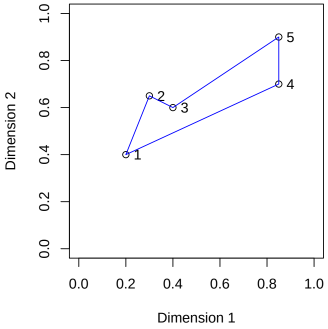


EXAMPLE: HIERARCHICAL CLUSTERING

Agglomerative hierarchical clustering starts with all points forming their own cluster and iteratively merges them until all points form a single cluster containing all points.

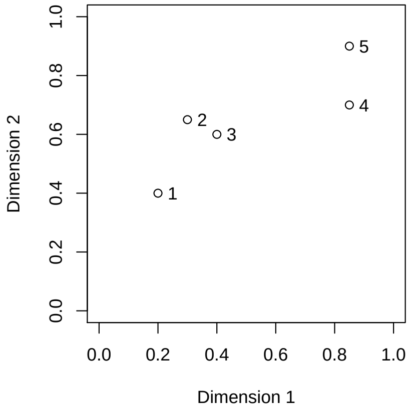
Example:

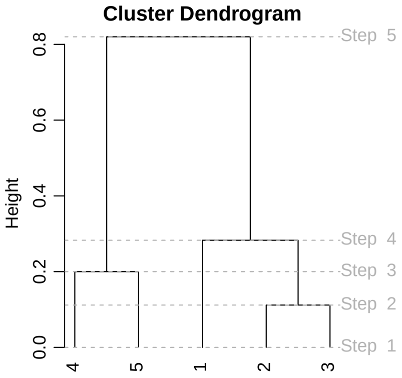
Step 1: $\{1\}, \{2\}, \{3\}, \{4\}, \{5\}$
Step 2: $\{1\}, \{2, 3\}, \{4\}, \{5\}$
Step 3: $\{1\}, \{2, 3\}, \{4, 5\}$
Step 4: $\{1, 2, 3\}, \{4, 5\}$
Step 5: $\{1, 2, 3, 4, 5\}$



DENDROGRAM

A Dendrogram is a tree showing which clusters / observations are merged after each step. The 'height' is proportional to the distance between the two merged clusters:

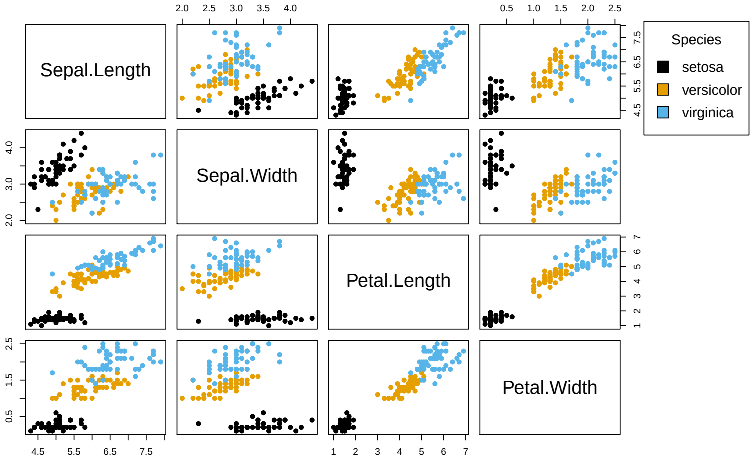




R EXAMPLE WITH IRIS DATA

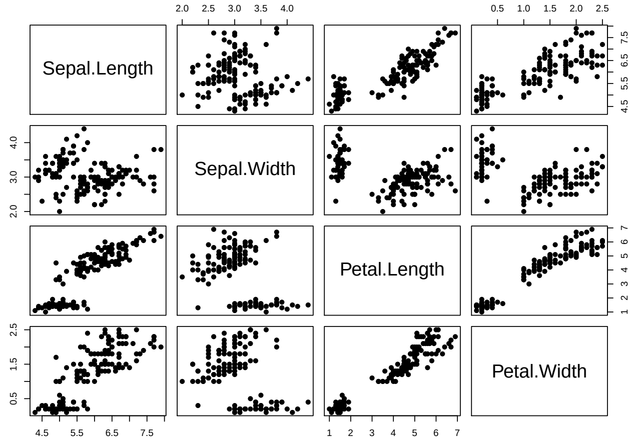
The data contains 150 leaf measurements for 3 flower species:

```
pairs(iris[1:4], col = iris$Species)
```



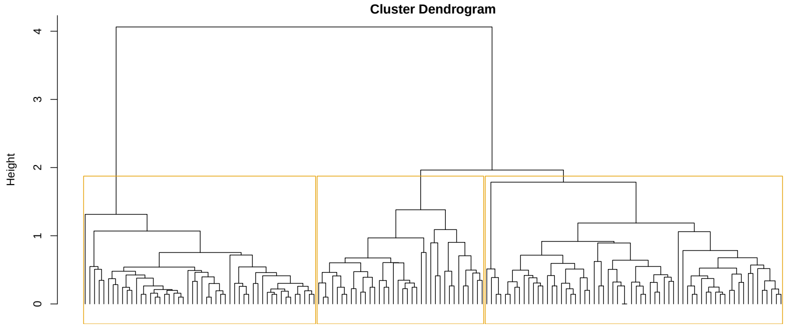
R EXAMPLE WITH IRIS DATA

We now "forget" the real groups specified by the ‘Species‘ variable and try to find clusters based on the leaf measurements.



R EXAMPLE WITH IRIS DATA

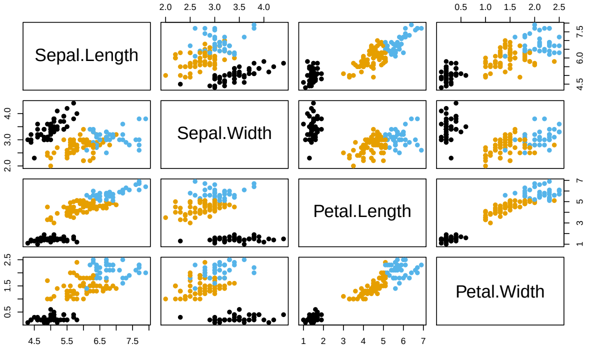
```r
# compute distance matrix
d.euclid = dist(iris[1:4], method = "euclidean")
# do clustering with average linkage
cl = hclust(d.euclid, method = "average")
plot(cl, labels = FALSE, hang = -1) # plot dendrogram
rect.hclust(cl, k = 3) # highlight the k = 3 groups
```



R EXAMPLE WITH IRIS DATA

We can extract the clustering assignments by cutting the dendrogram, e.g. using $k = 3$ clusters:

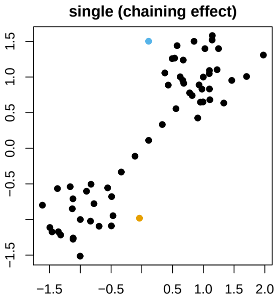
```
group = cutree(cl, k = 3) # get clusters assignments for k=3
pairs(iris[1:4], col = group) # plot clusters with different colors
```

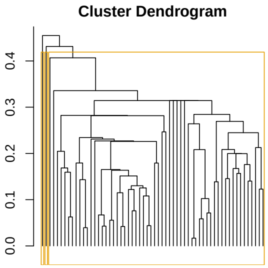


PROPERTIES: SINGLE LINKAGE

Single linkage introduces the *chaining problem*:

- Only one pair of points needs to be close to merge clusters.
- A chain of points can expand a cluster over long distances.
- Points within a cluster can be too widely spread and not dense enough.

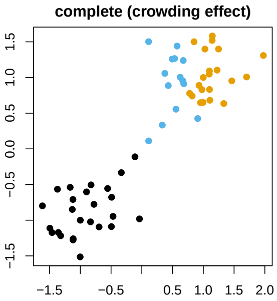


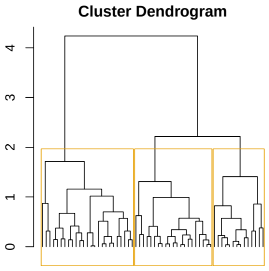


PROPERTIES: COMPLETE LINKAGE

Complete linkage avoids chaining, but suffers from *crowding*:

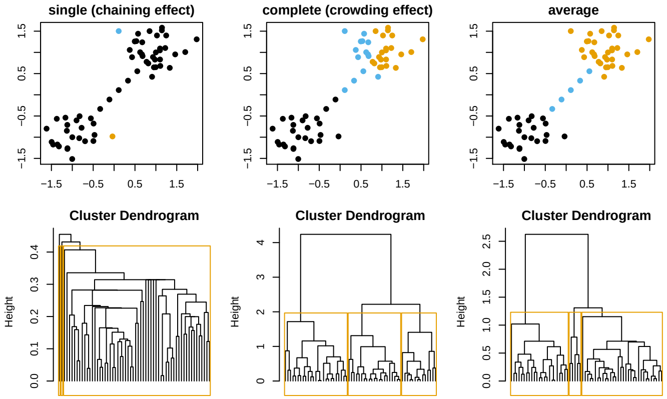
- Merging is based on the furthest distance of point pairs from different clusters.
- Points of two different clusters can thus be closer than points within a cluster.
- Clusters are dense, but too close to each other. <!– and sensitive to outliers. –>





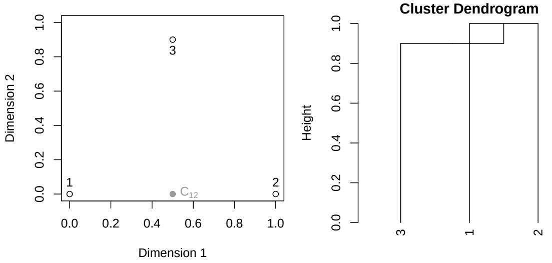
PROPERTIES: AVERAGE LINKAGE

- Average linkage is based on the average distance between clusters and tries to avoid crowding and chaining.
- Produces clusters that are quite dense and rather far apart.



PROPERTIES: CENTROID LINKAGE

- Centroid linkage defines the distance based on **artificial data points** (the cluster centers), which produces dendrograms **with inversions**, i.e., the distance between the clusters to be merged can be smaller in the next step.
- In single, complete and average linkage, the distance between the clusters to be merged increases in each step. $\Rightarrow$ always produces dendrograms **without inversions**.



SUMMARY

- *Hierarchical agglomerative clustering methods* iteratively merge observations/clusters until all observations are in one single cluster.
- Results in a hierarchy of clustering assignments which can be visualized in a *dendrogram*. Each node of the dendrogram represents a cluster and its ‘height’ is proportional to the distance of its child nodes.
- The most common linkage functions are *single, complete, average* and *centroid* linkage. There is no perfect linkage and each linkage has its own advantages and disadvantages.

Partitioning Clustering Methods

OPTIMAL PARTITIONING CLUSTERING

Hierarchical clustering:
Stepwise merging (agglomerative methods) or dividing (divisive methods) of clusters based on distances and linkages. The number of clusters are selected by splitting the dendrogram at a specific threshold for the “height” after visual inspection.

Partitioning clustering:
Partitions the N observations into a predefined number of K clusters by optimizing a numerical criterion. The most common partitioning methods are:

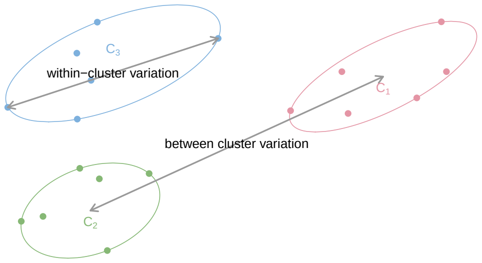
- K-means
- K-medians
- K-medoids (Partitioning Around Medoids (PAM))

K-MEANS

K-means partitions the N observations into K predefined clusters $C_1, C_2, \ldots, C_K$ by minimizing the **compactness**, i.e. the **within-cluster variation** of all clusters using

$$\sum_{k=1}^{K} \sum_{i \in C_k} \|X_i - \bar{X}_k\|_2^2 \to \min,$$

where $\bar{X}_k = \frac{1}{N_k} \sum_{i \in C_k} X_i$ is the centroid of cluster k and N_k is the number of observations in cluster k.



K-MEANS

Idea: Consider every possible partition of N observations into K clusters and select the one with the lowest **within-cluster variation**.
Problem: Requires trying all possible assignments of N observations into K clusters, which in practice is nearly impossible (Hothorn et al., 2009, p.322):

N	K	Number of possible partitions
15	3	2.375.101
20	4	45.232.115.901
100	5	10^{68}

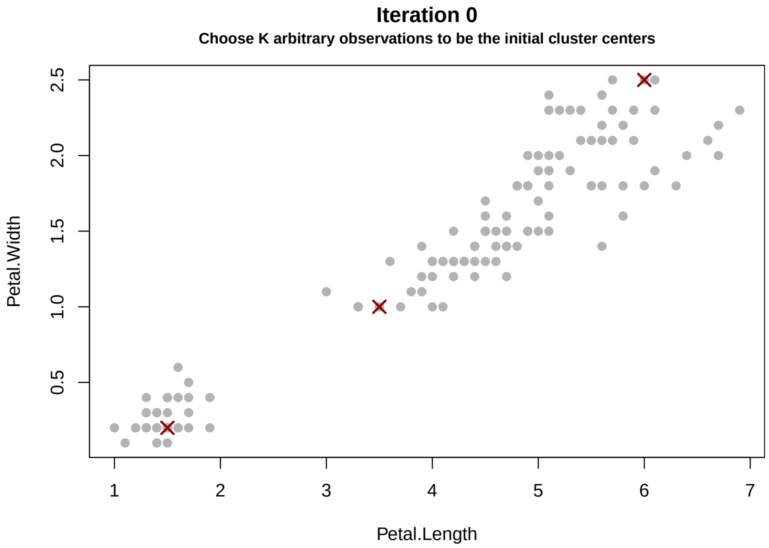
Hothorn, T., Everitt, B. S. (2009). A handbook of statistical analyses using R. Chapman and Hall/CRC.

K-MEANS

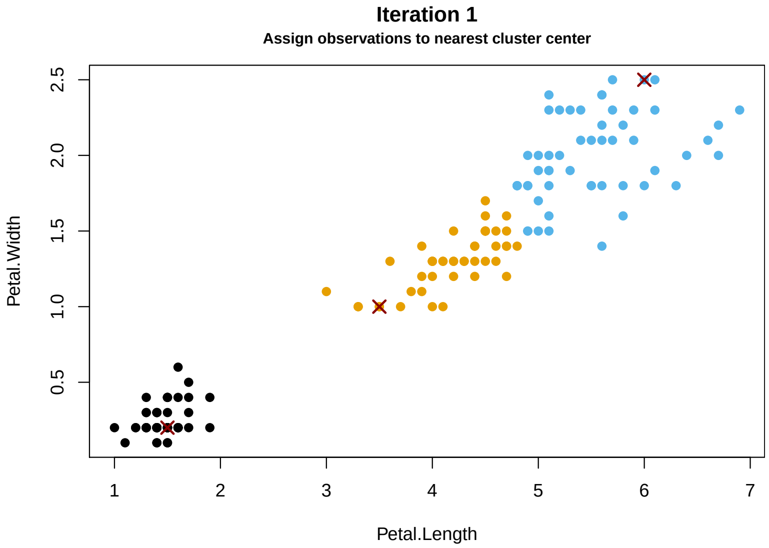
Use an approximation:

1. **Initialization:** Choose K arbitrary observations to be the initial cluster centers.
2. **Assignment:** Assign every observation to the cluster with the closest center.
3. **Update:** Compute the new center of each cluster as the mean of its members.
4. Repeat (2) and (3) until the centers do not move.

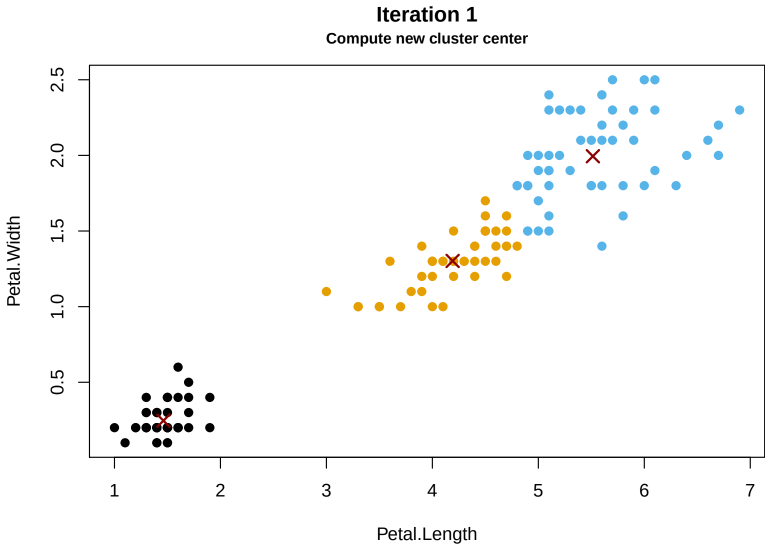
K-MEANS EXAMPLE



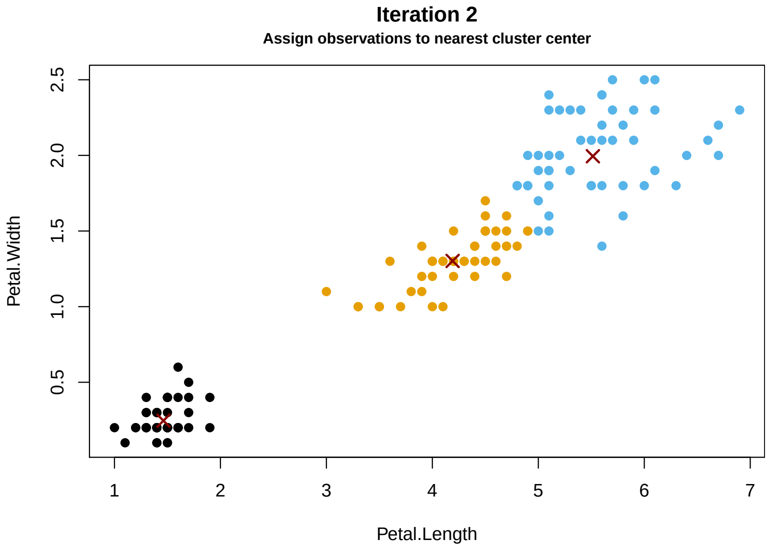
K-MEANS EXAMPLE



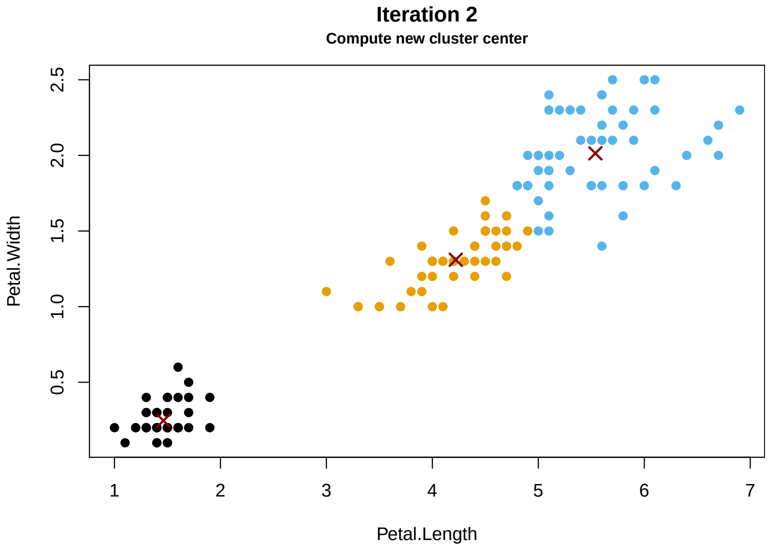
K-MEANS EXAMPLE



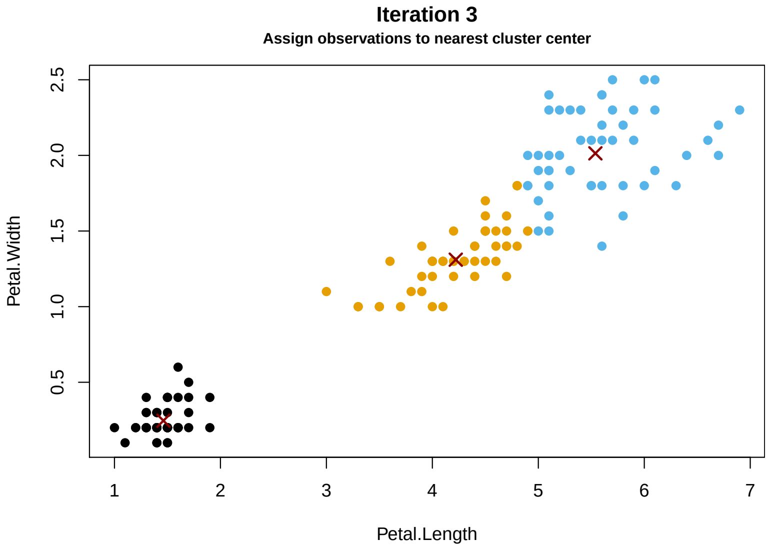
K-MEANS EXAMPLE



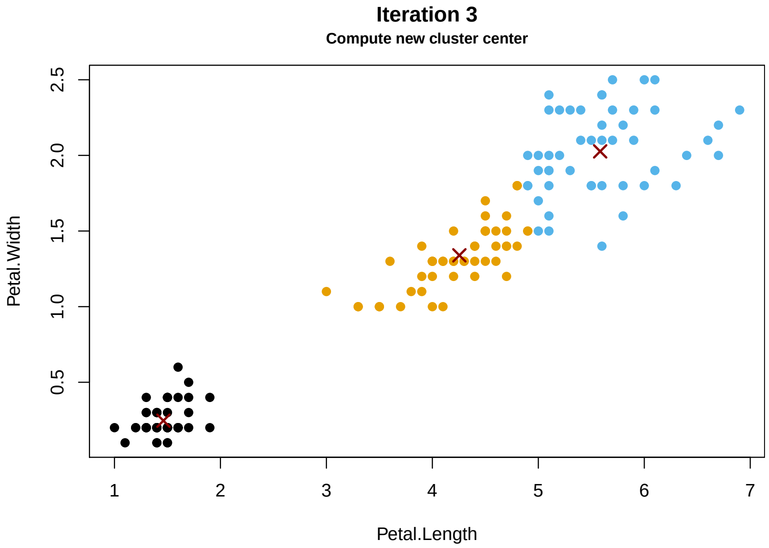
K-MEANS EXAMPLE



K-MEANS EXAMPLE



K-MEANS EXAMPLE



K-MEANS IN R

The *K*-means algorithm is part of the base distribution in R, given by the “kmeans” function (using “algorithm = "Lloyd"”):

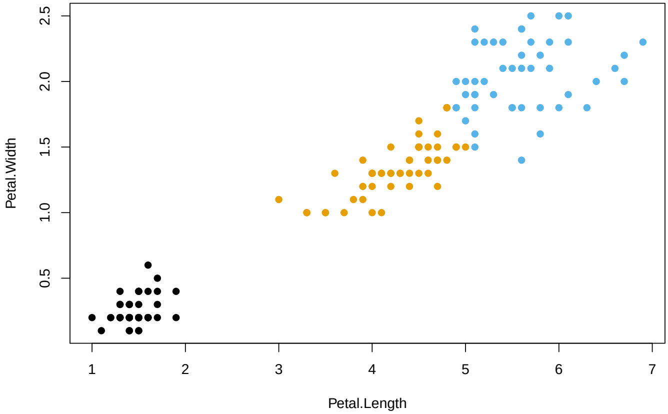
```
km = kmeans(iris[,3:4], centers = 3, nstart = 100,
  iter.max = 100, algorithm = "Lloyd")
str(km)

## List of 9
##  $ cluster     : int [1:150] 1 1 1 1 1 1 1 1 ...
##  $ centers     : num [1:3, 1:2] 1.462 4.269 5.596 0.246 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:3] "1" "2" "3"
##   .. ..$ : chr [1:2] "Petal.Length" "Petal.Width"
##  $ totss       : num 551
##  $ withinss    : num [1:3] 2.02 13.06 16.29
##  $ tot.withinss: num 31.4
##  $ betweenss   : num 520
##  $ size        : int [1:3] 50 52 48
##  $ iter        : int 17
##  $ ifault      : NULL
##  - attr(*, "class")= chr "kmeans"
```

K-MEANS IN R

The final cluster assignments can be visualized by:

```r
plot(iris[,3:4], pch = 19, col = km$cluster)
```

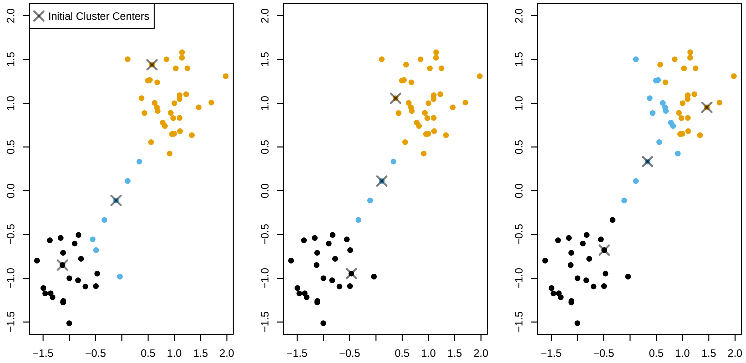


PROPERTIES OF *K*-MEANS

- *K*-means is based on computing the mean, which is sensitive to outliers and can only be computed for numerical data.
- The **within-cluster variation** is reduced in each iteration. In R, the maximum number of iterations is specified by `iter.max`.
- The final result is typically not the best result that globally minimizes the **within-cluster variation**.
  $\rightarrow$ would only be possible after trying all possible partitions!
- *K*-means can be restarted multiple times using `nstart`. The clustering with the smallest within-cluster variation is then selected as the best solution.

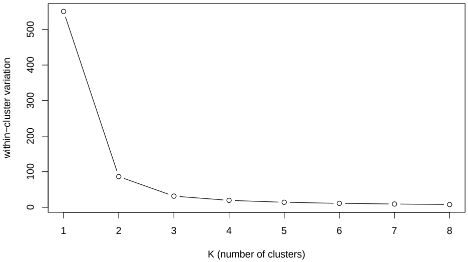
PROPERTIES OF *K*-MEANS

- *K*-means produces different clusters depending on the initial centers and always converges, e.g.:



CHOICE OF K

- Many methods exist for choosing the number of clusters K (there is no perfect solution).
- The easiest method is to apply K-means for different K and plot the **within-cluster variation** for each number of K.
- The **within-cluster variation** always decreases with increasing number of clusters.
- An **"elbow"** in the plot might indicate a useful solution.



K-MEDOIDS

- is strongly related to K-means and is realized by the **Partitioning Around Medoids (PAM)** algorithm.
- uses cluster medoids as representative clusters, i.e. **real data points** instead of **artificial data points** (such as the cluster centers as in K-means) are used.
- is less sensitive to outliers and more robust than K-means.
- can handle categorical features (K-means doesn't because it is based on calculating the cluster centers by taking the mean in each dimension).

THE PAM ALGORITHM

1. **Initialization:** Randomly select K data points as the medoids.
2. **Assignment:** Assign each data point x_i to its closest medoid m and calculate the within-cluster variation for each medoid (by summing up the distances of the current medoid m to all other data points associated to m) .
3. **Update:** Swap m and x_i and recompute the within-cluster variation to see if another medoid is more appropriate. Select the medoid m with the lowest within-cluster variation.
4. Repeat steps (2) and (3) until medoids do not change.

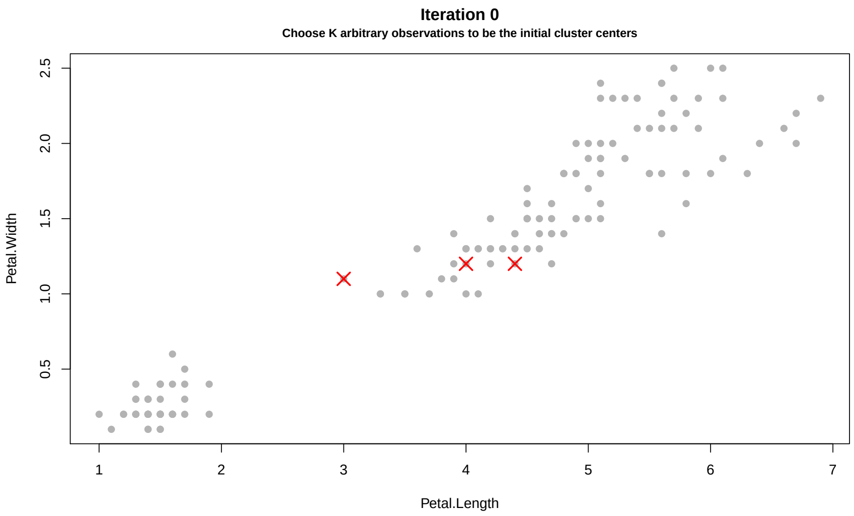
THE PAM ALGORITHM

The PAM algorithm typically uses the following two metrics to compute distances:

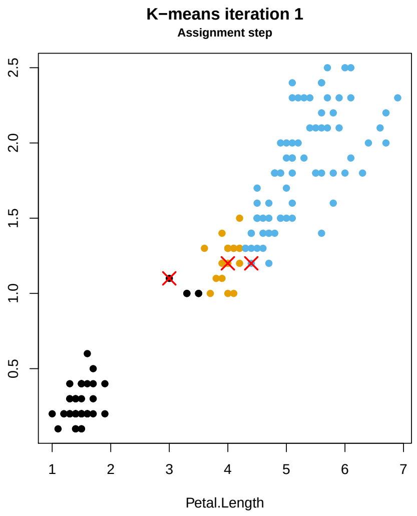
- The euclidean distance (root sum-of-squares of differences).
- The Manhattan distance (the sum of absolute distances).

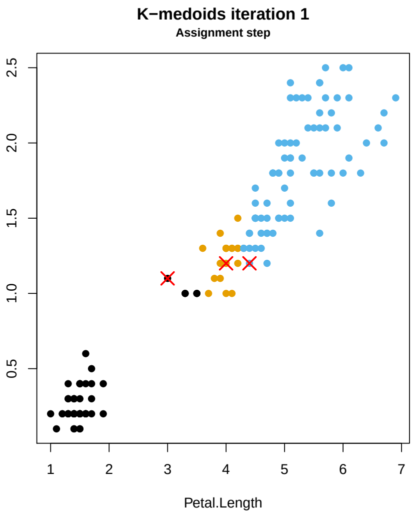
Note: The Manhattan distance should give more robust results if your data contains outliers. In all other cases, the results will be similar for both metrics.

K-MEANS VS. K-MEDOIDS

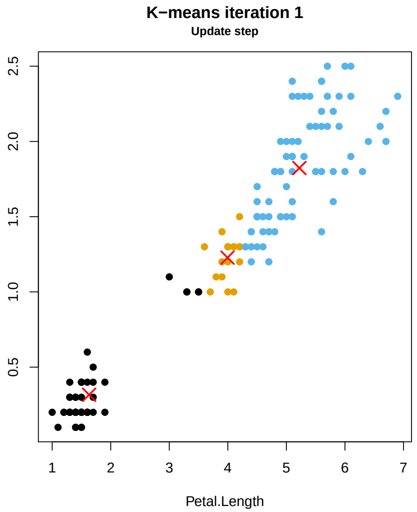


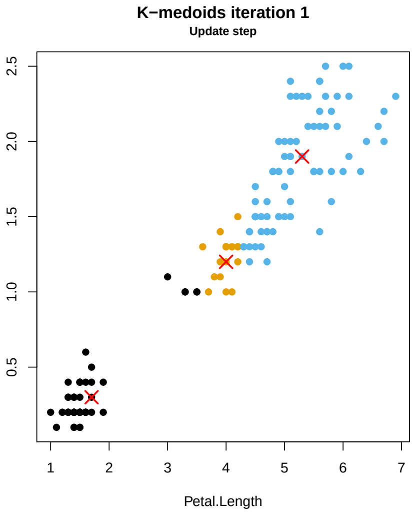
K-MEANS VS. K-MEDOIDS



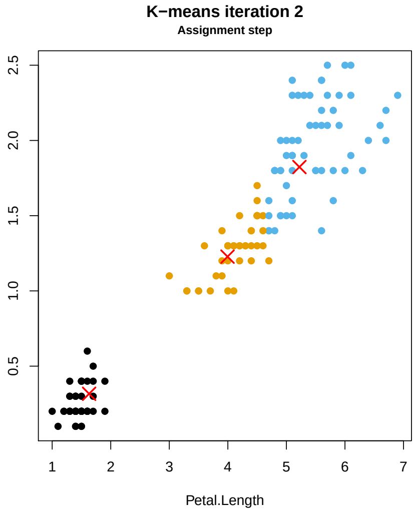


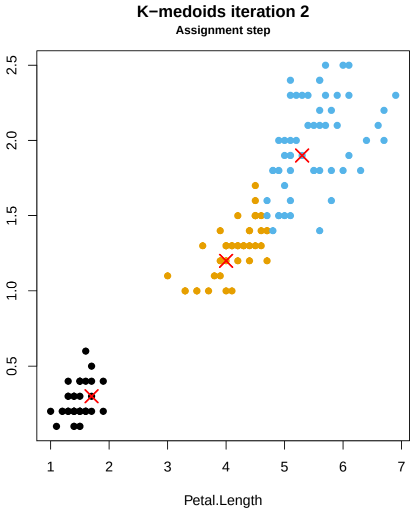
K-MEANS VS. K-MEDOIDS



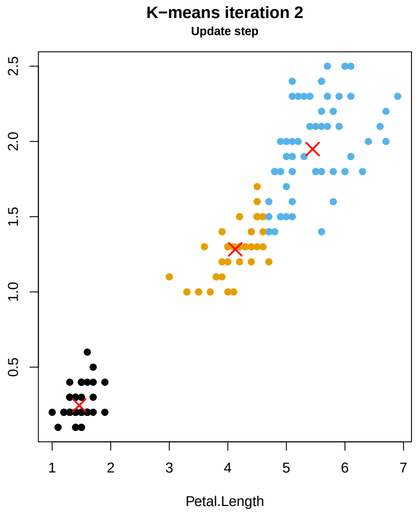


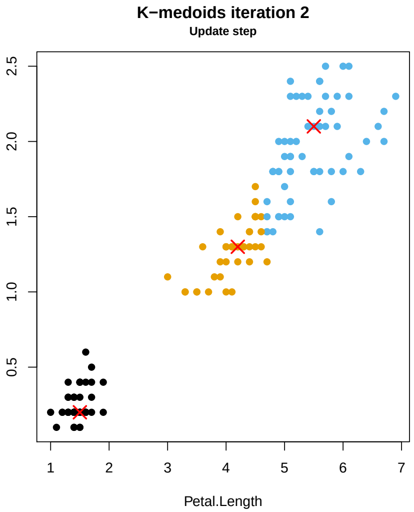
K-MEANS VS. K-MEDOIDS





K-MEANS VS. K-MEDOIDS





K-MEDOIDS IN R

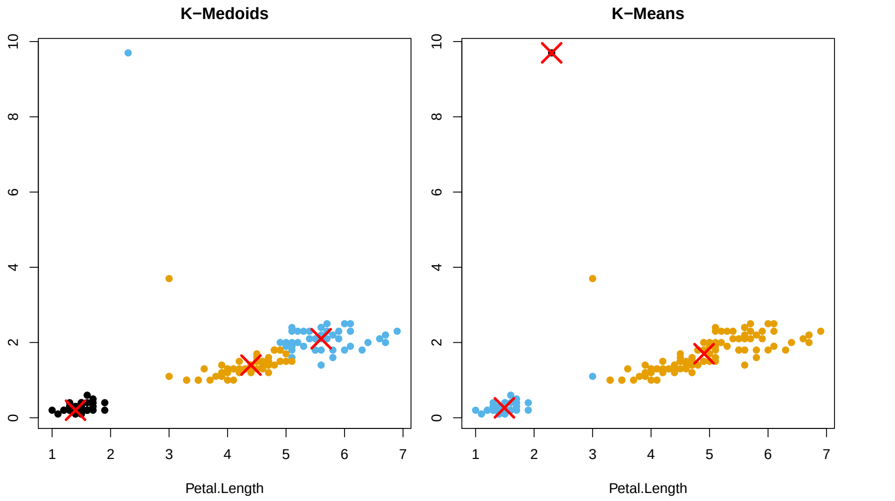
The *K*-medoids algorithm is implemented in the ‘pam‘ function included in the ‘cluster‘ R-package.

We compare *K*-means and *K*-medoids on a modified iris data with additional outliers:

```r
library(cluster)
# add outliers to iris
out = data.frame(Petal.Length = c(2.3, 3), Petal.Width = c(9.7, 3.7))
iris2 = rbind(iris[, 3:4], out)

# K-medoid vs. K-means
kmedoid = pam(iris2, k = 3)
km = kmeans(iris2, centers = 3, iter.max = 100, nstart = 100)
```

K-MEDOIDS IN R



SUMMARY

- Minimizing the *within-cluster variation* exactly is not feasible and can be approximated by the K-means algorithm.
- K-means always converges, however, the cluster assignments strongly depend on the initial centers.
  $\rightarrow$ repeat it several times with different initial centers.
- A simple solution for choosing the number of clusters K is to plot the *within-cluster variation* for several K and look for an *"elbow"* which is a good guess for K.