

Clustering

Gilles San Martin

01 October 2018 - 13h08

Contents

1	Clustering	2
1.1	Clustering aims and general process	2
1.2	Different clustering algorithms	3
1.3	Hierarchical (agglomerative) clustering	5
1.3.1	Introduction	5
1.3.2	Details about the algorithm and different grouping methods	9
1.3.3	Choice of the grouping method	13
1.3.4	Flexible-Beta clustering	22
1.3.5	Interpretation of the dendrogram and common pitfalls	24
1.3.6	Basic dendrograms manipulation and graphs	25
1.4	K-means and K-medoids partitioning	34
1.5	Clusters interpretation	37
1.5.1	Clusters visualisation	37
1.5.1.1	Visualize the clusters with heatmaps	37
1.5.1.2	Visualize the clusters on a SPLOM	45
1.5.1.3	Visualize the clusters on an ordination plot	46
1.5.2	Clusters description	48
1.5.2.1	Describe the clusters with simple graphs	48
1.5.2.2	Describe the clusters with pseudo-supervised approaches (eg : classification tree)	52
1.6	Clustering validation	54
1.6.1	Silhouette width	55
1.6.2	Choose the “best” number of clusters with the Gap statistic	58
1.6.3	Choose the “best” number of clusters with NbClust	60
1.6.4	Compare different algorithms and different number of clusters with the silhouette width and Dunn index	62
1.6.5	Compare different algorithms and different number of clusters with clValid	66

1 Clustering

```
source("/home/gilles/stats/mytoolbox.R")
setwd("/home/gilles/stats/Formation_R_stats/Formation_Stats_4_Multivariate/")

# load the packages used for this chapter
library(vegan)
library(gplots)
library(dendextend)
library(visreg)
library(multcomp)
library(rpart)
library(rpart.plot)

# load ggplot, change the default theme and change the locale (language = English)
library(ggplot2)
```

1.1 Clustering aims and general process

The aim of clustering is to create discrete groups (called “clusters”) of observations (or variables) that are as similar as possible within the cluster but as different as possible from the other clusters.

There are two main (non exclusive) reasons for which you may want to create such groups :

1. **Data visualisation** : To reorganize the data in a way that helps you to visualize general patterns in it. Clustering is then a data exploration tool used mainly for visualisation (eg dendrograms and heatmaps). The clusters created will probably not be used outside your specific study.
2. **Data simplification** : To simplify the data in a few easily understandable groups. In that case you want to create a “typology” or “classification” that might be reused outside your study (ex : phyto-sociological associations, geomorphological regions, land-use typologies for mapping, ...). In this case, assessing the validity of the clusters is probably more important while for data exploration, any clustering that provides interpretable results might be useful.

All processes studied in biology are not always really discrete. For example plant species communities might be very different between a peat bog and a chalk grassland but there are probably many intermediate plant communities that fall between the peat bogs and chalk grasslands. However even for rather continuous cases like this one it might be useful to simplify reality by creating a typology/classification of “typical” plant associations.

We will divide the clustering approach in several steps :

1. **Clustering construction** : Choose a clustering algorithm and an appropriate distance metric if needed
2. **Clustering visualisation** and interpretation : What is the meaning of these clusters ? What characterise them ?
3. **Clustering validation** : assess the quality of the clusters to answer the following questions :
 1. How “good” are my clusters ?
 2. How many clusters should I use ?
 3. Which clustering method works “best” ?
 4. Are the clusters “real” or am I creating clusters from random noise ?

This is not a linear and straightforward process... Clustering interpretation is by far the most important step and might help to choose for example the clustering algorithm (ie one that produce interpretable results is better...) and the number of clusters. External validation is also often used to help the interpretation of the clusters. Clustering validation and interpretation are generally particularly interleaved processes...

1.2 Different clustering algorithms

There are many different clustering algorithms but the most popular is by far a group of hierarchical agglomerative clustering methods (eg `hclust` function in base R or `cluster::agnes`). When one claims to use “clustering” without more precision on the method you can safely assume that this approach was used. Another very popular approach particularly for huge datasets is k-means clustering (and the related PAM algorithm).

These clustering algorithms are often classified according to certain technical characteristics :

- **Agglomerative vs divisive :**

Agglomerative algorithms like `hclust` start by grouping similar points then progressively agglomerate them into larger clusters.

Divisive algorithms (like `kmeans` or `cluster::diana`) start with the whole dataset and split it in several groups.

- **Hierarchical or not :**

In hierarchical clustering (like `hclust`) clusters are subdivided into subclusters and form a hierarchy of clusters that is generally represented graphically with a dendrogram. The user can decide afterwards how many clusters will be interpreted or used. When you go from a two clusters solution to a 3 clusters solution one of the two clusters is divided into 2 clusters and the other is left unchanged. In non-hierarchical methods (like `kmeans`) the user decides in advance how many clusters should be found and the clusters do not show any hierarchical relationship between each other. When you go from a 2 clusters solution to a 3 clusters solution an entirely new solution is computed and the original 2 clusters points might end up in different clusters.

- **Fuzzy vs Crisp/Hard :** In “crisp” or “hard” clustering (like `hclust` and `kmeans`) one point can only be the member of one cluster.

In fuzzy clustering methods (like `cluster::fanny`) we estimate for each point a probability to be a member of each cluster. From this result one can obtain a hard clustering by placing each point in the cluster for which it has the highest probability or decide to consider some points as “outliers” or “noise” when their cluster membership is unclear/

- **explicit vs implicit distance :** for most algorithms, you must provide a distance matrix of your choice. But some algorithms

The combination of these characteristics can help to have a vague idea of how different algorithms work and what are their similarities/dissimilarities

- Hierarchical Clustering (`hclust`, `cluster::agnes`): hierarchical, agglomerative, crisp on any distance matrix
- DIANA (`cluster::diana`) : hierarchical, divisive, crisp on any distance matrix
- K-means (`kmeans`, `vegan::cascadeKM`) : non hierarchical, divisive, crisp - based on Euclidean distance (or a few compatible distances via transformation : Hellinger, Chord, ...)
- PAM (“Partitioning Around Medoids” or “k-medoids”) and CLARA (“Clustering Large Applications”) (`cluster::pam`, `cluster::clara`) : non hierarchical, divisive, crisp - based on any distance matrix. PAM is quite slow. For larger datasets CLARA applies PAM on random subsample of the total dataset and provide a global solution in a faster algorithm.
- FANNY (`cluster::fanny`) fuzzy c-means clustering : non hierarchical, divisive, fuzzy - based on any distance matrix

Many other algorithms exist and might not necessarily fit this typology or they might have a very specific application...

- DBSCAN ("Density Based Clustering" `dbscan::dbscan`) : designed specifically to find dense clusters of points of any shape while most other methods will tend to find circular shapes. Two other very attractive features of this algorithm is that the number of clusters is found automatically and some points will also be detected as "outliers" and will not be placed in any cluster. However it needs dense clusters and hence might not work well on highly dimensional datasets.
- Model based clustering (`mclust::mclust`) : a sort of non hierarchical fuzzy method in which each cluster is considered to be a multivariate distribution (Normal, Poisson, Negative-Binomial,...). The parameters of these distributions are estimated by Maximum Likelihood with the EM algorithm ("Expectation-Maximisation") or other similar algorithms (eg Deterministic or Simulated Annealing - DA - SA). The "best" model and number of clusters is automatically chosen based on the maximisation of the BIC criterion. For each point we obtain a probability to be in each cluster (fuzzy clustering). Quite popular in genomics.
- SOM (Self Organizing Maps `kohonen::som`) : based on Neural Networks and euclidean + a few other distances. Can handle mixtures of quantitative and qualitative data and also missing data. The clusters are spatially organized in one or 2 dimensions. As a consequence the method is often used at the same time as a clustering method and an ordination/dimensionality reduction method.
- SOTA (Self-Organizing Tree Algorithm - `clValid::sota`) : hierarchical, divisive, crisp - only for variable clustering (R mode) with euclidean distance or Pearson correlation (mainly developed for genomic datasets)
- k-modes (`klaR::kmodes`): similar to k-means (non hierarchical, divisive, crisp) but specifically developed to handle qualitative data

1.3 Hierarchical (agglomerative) clustering

1.3.1 Introduction

In hierarchical (agglomerative) clustering the observations are progressively grouped in larger clusters and the hierarchical structure is typically represented with a dendrogram (tree like representation of the hierarchy).

The typical basic steps are

1. compute an appropriate distance (`dist`, `vegan::vegdist`) sometimes after an appropriate transformation of the data (scaling, log, hellinger,...)
2. apply the clustering algorithm (`hclust`) and choose a grouping method (method = average, ward.D,...)
3. visualise the result as a dendrogram (`plot`)
4. cut the dendrogram at a given height to attribute each observation to a given cluster (`cutree`)

These are only the basic steps. The interpretation of the clusters (and to some extent their “validation”) is of tremendous importance. Producing a dendrogram without being able to interpret the clusters formed is often useless.

Example of the basic steps =

```
# Create a fake dataset with 20 observations * 5 variables
set.seed(123)
d <- rbind( matrix(runif(50, 0, 1), 10, 5),
            matrix(runif(25, 1, 2), 5, 5))
colnames(d) <- paste0("x", 1:5)
# scale (standardize) the data (for each column subtract the average and divide by the
# standard deviation) and compute the Euclidean distance
Eucl_dist <- dist(scale(d))
# compute the agglomerative clustering with a "ward.D2" method for grouping
hcl <- hclust(Eucl_dist, method = "ward.D2")
# Visualize the dendrogram
plot(hcl, hang = -1)
```

Cluster Dendrogram

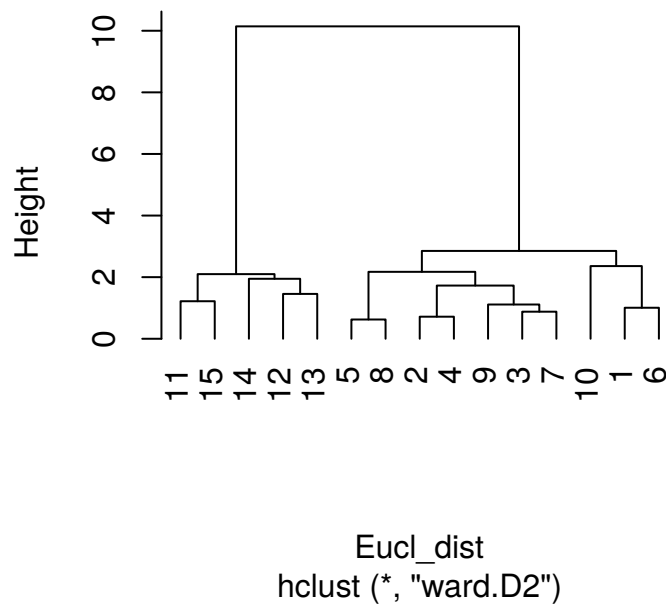


Figure 1:

```
# Cut the clustering solution to obtain 2 groups
# Here the first ten observations are in cluster nr1 and the 5 next ones are in cluster
# 2 (this is how the dataset has been created)
cutree(hcl, k = 2)
```

```
## [1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2
```

You can also apply clustering on the variables providing that you use an appropriate distance method. Reminder : the distance functions generally work on the lines so if you want to apply them on the columns you will need first to transpose your dataset (function `t()`). Note that here the 5 variables are completely independent so there is no real cluster in the variables...

```
# Compute the euclidean distance on the columns :
colDist <- dist(t(scale(d)))
# hierarchical clustering with an "average" (UPGMA) grouping method
hcl <- hclust(colDist, method = "average")
# Visualize the results with a dendrogram
plot(hcl, hang = -0.01)
```

Cluster Dendrogram

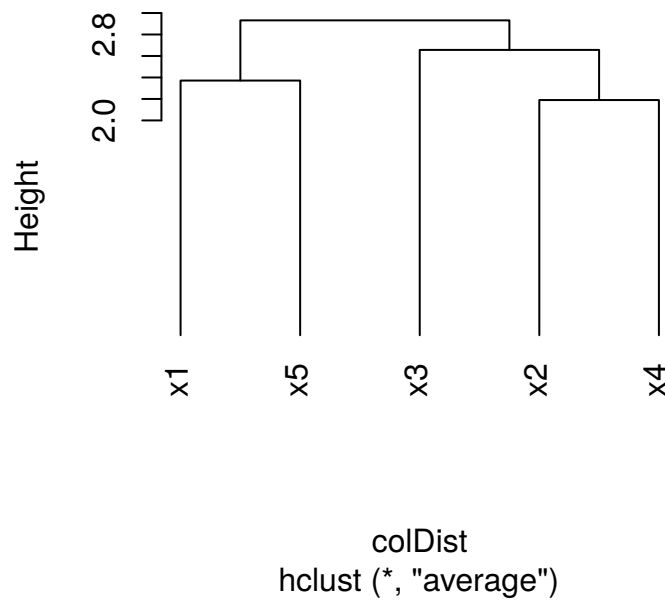


Figure 2:

Advantages

- Provide a hierarchical structure that is generally helpful to organise different levels of grouping structures/classification
- Easy to visualise with dendrograms and heatmaps
- You don't need to specify in advance how many clusters you will use. You can decide visually on the dendrogram (+heatmaps, +external validation) how many clusters you want to use
- In contrast with ordinations, all original dimensions are represented at once

Disadvantages

- visualisation with dendrograms might be difficult for very large datasets
- in contrast with non agglomerative clustering, once an observation is attributed to one cluster, its cluster membership will no more be reevaluated (and hence there is no possibility to "correct" wrong assignments)
- the choice of the grouping method is rather arbitrary and provides very different results particularly at the top levels of the dendrogram (because of the agglomerative process) which are generally the ones that are the main focus
- Not easy to reattribute a new observation to the clusters created without rebuilding the whole dendrogram

Recommendations

- When you use Hierarchical clustering, always specify :
 1. any data transformation that you have applied
 2. which distance index you have chosen (and choose it wisely...)
 3. which agglomeration method you use
 4. If you are clustering the observations or the variables (although this might be self-evident from the rest)
- Give a verbose description of the groups created with the help of any interpretation method : graphs of the original variables per group, heatmaps, external supplementary variable,...

- don't believe blindly the cluster validation indices unless you cannot do otherwise

1.3.2 Details about the algorithm and different grouping methods

The algorithm works as follow :

- compute the distance between all observations
- group first the observations with the lowest distance
- recompute the distance between all individual observations and grouped observations
- aggregate the ones with the lowest distance etc...

The first step is quite straightforward (once you have chosen the right distance index). The difficulty comes when you want to measure the distance between two groups of observations or between one observation and a group of observations. There are many different ways to do so that can be chosen via the `method` argument of `hclust` (see below).

NB : see Legendre and Legendre (2012) if you want to understand exactly how the distances between clusters are recomputed. However this is not of major importance for the practical use of these algorithms.

- **single** : single linkage : The distance between 2 clusters is the minimum distance between their members. Single Linkage will put more weight on between-cluster separation at the expense of within-cluster homogeneity. It tends to produce “chains” (‘friends of friends’ clustering strategy) : once a cluster is formed, the algorithm tends to add the new observations to this cluster, one by one instead of starting a new cluster.
- **complete** : = complete linkage (default). The distance between 2 clusters is the maximum distance between their members. When you have two large clusters the probability that at least two of their members are far away from each other is higher and it is difficult to join existing clusters. This method tends to produce more spherical, equally sized clusters which is often useful. However, in single and complete linkage the distance between two clusters is based on a single pair of observations. These methods are therefore quite sensitive to outliers. Note also that transformations keeping the order of the data (eg sqrt, log) will not decrease the influence of these outliers and they will provide exactly the same tree structure (this is not the case with the next methods).
- **average** or “UPGMA” (“Unweighted pair group method with arithmetic mean”). The distance between 2 clusters is the average of the distances between their members. So the distance between two clusters is based on all pairs of observations and this method is less sensitive to outliers. The dendrograms based on this method also provide generally the best approximation of the distance between two observations. It provides generally intermediate results between complete and single linkage, with less chaining than single linkage and more irregular clusters than complete linkage. This is a very widely used method (typically used for example in phylogeny).
- **mcquitty** = “WPGMA” (Weighted Pair Group Method with Arithmetic Mean). This method is close to UPGMA but each of the clusters joined will have the same contribution to the computation of the average whatever the number of observations in each cluster.
- **centroid** = ‘UPGMC’ and **median** = ‘WPGMC’ are using the centroid of the clusters to compute the distances (with a weighted or unweighted algorithm taking into account the difference of cluster size or not). These methods can produce “inversions” on the dendrograms that are not easy to interpret (the node of a child cluster might have a higher value than the node of its parent cluster).
- **ward.D** and **Ward.D2** : the ward method will choose to group clusters that minimise the within cluster distance. With Euclidean distance this creates clusters that minimise the within group variance. This method is very popular because it tends to create well separated uniform groups and make dendrograms in which the major groups are easy to visualize. However the smaller distances are generally rather compressed which make the visualisation and interpretation of clusters of more similar observations more difficult to read. The 2 methods are two versions of the same algorithm : the dissimilarities are squared before cluster updating for ward.D2 while they are not for ward.D

The following very simple example (one dimensional dataset) shows how the clustering works and how the

dendrograms should be read. (NB : in real life applying clustering or ordination on one or two dimensional datasets is really unnecessary and must - most of the time - be avoided).

```
d <- cbind(x = c(1, 2, 4, 8, 10), y = 0)
row.names(d) <- c(1, 2, 4, 8, 10)

# dev.new(width = 10/2.54, height = 4/2.54)
par(mfrow = c(1,1), mar = c(3.5,3.5,1,1), mgp = c(2, 0.6, 0), cex = 0.8, las = 1)
plot(d, type = "n")
text(d, labels = d[, "x"])
```

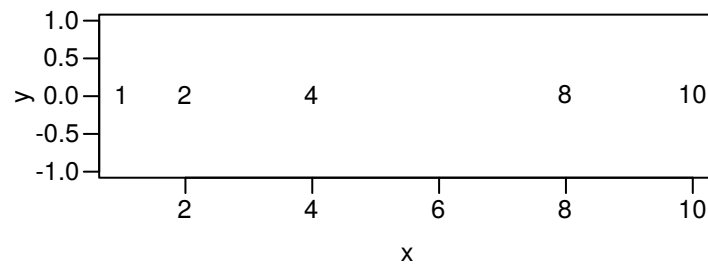


Figure 3:

The Euclidean distance is very easy to compute even without computer here :

```
D <- dist(d)
D

##      1 2 4 8
## 2    1
## 4    3 2
## 8    7 6 4
## 10   9 8 6 2
```

You can see on these dendrograms that :

- observations 1 and 2 are always at a distance of 1 (position of the node)
- observations 8 and 10 are always at a distance of 2
- the different grouping methods differ only on the position of the node linking observation 4 to the cluster of observation 1 and 2 : 2 for single linkage, 3 for complete linkage, 2.5 for average, etc...

Note also that here all the graphs are very similar but on real life datasets these different grouping methods will generally provide very different solutions...

```

# dev.new(width = 14/2.54, height = 20/2.54)
xhang = -1

par(mfrow = c(4,2), mar = c(0.1, 2.5, 2, 1), mgp = c(1.5, 0.5, 0))
# Single : shortest distance among groups
plot(hclust(D,method = "single"), hang = xhang , ylab="", main = "Single linkage")
abline(h = seq(0,10,1), col = "gray70")

# Complete : longest distance among groups
plot(hclust(D,method = "complete"), hang = xhang , ylab="", main = "Complete linkage" )
abline(h = seq(0,10,1), col = "gray70")

# Average : average distance among groups = UPGMA
plot(hclust(D,method = "average"), hang = xhang , ylab="", main = "Average = UPGMA" )
abline(h = seq(0,10,1), col = "gray70")

# WPGMA
plot(hclust(D,method = "mcquitty"), hang = xhang , ylab="", main = "Mc Quitty = WPGMA" )
abline(h = seq(0,10,1), col = "gray70")

# UPGMC
plot(hclust(D,method = "centroid"), hang = xhang , ylab="", main = "Centroid = UPGMC" )
abline(h = seq(0,10,1), col = "gray70")

# Median = WPGMC
plot(hclust(D,method = "median"), hang = xhang , ylab="", main = "Median = WPGMC" )
abline(h = seq(0,10,1), col = "gray70")

# Ward 1
plot(hclust(D,method = "ward.D"), hang = xhang , ylab="", main = "Ward 1" )
abline(h = seq(0,10,1), col = "gray70")

# Ward 2
plot(hclust(D,method = "ward.D2"), hang = xhang , ylab="", main = "Ward 2" )
abline(h = seq(0,10,1), col = "gray70")

```

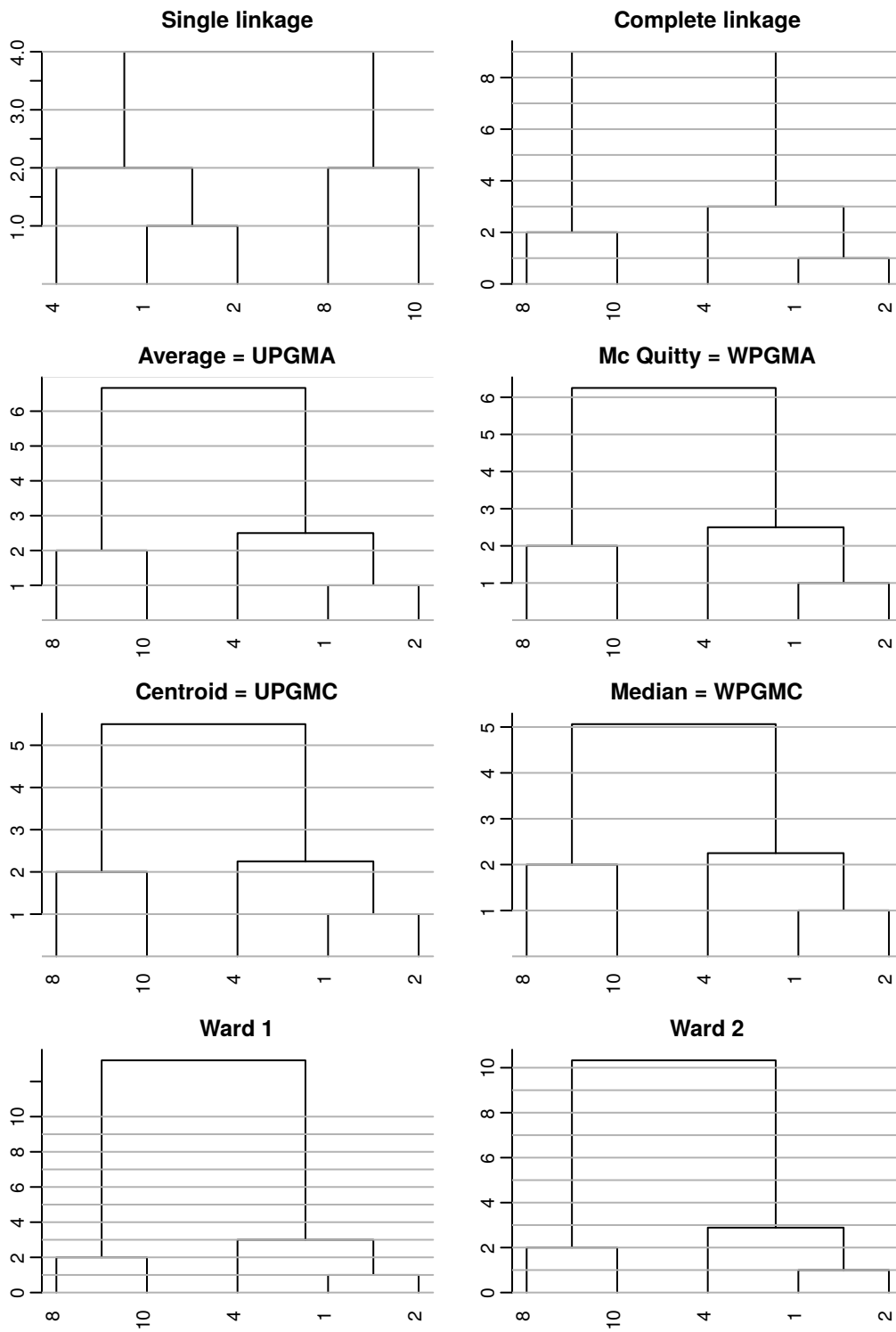


Figure 4:

1.3.3 Choice of the grouping method

These different methods will generally produce very different results and the choice of the grouping method is mostly arbitrary... The “best” grouping method will depend on your dataset and on your definition of a “good” cluster. So the most important criterion to choose the “best” method is to look at the clusters and dendrograms and choose the one that provides the best interpretable/useful result. The UPGMA (average), ward and complete linkage methods are popular choices that will produce well separated major groups. In community ecology (clustering of species data) “flexible-beta clustering” (with a beta of -0.25) has also become recently popular (see below). It provides an intermediate solution between complete linkage and UPGMA (average). In the validation section we will see descriptive statistics that can be computed to evaluate the “quality” of the clustering solution. However interpretability should always be your main driver to choose the method.

We will first examine how these different methods behave with completely random data (where there is actually no clusters...).

- All the methods (excepted maybe the single linkage) will produce clusters even in random noise where there are absolutely no clusters...
- all algorithms produce rather different solutions ...
- the single linkage tend to produce one (or few) large cluster and a few very small clusters of points that are weak outliers here
- The 2 ward algorithms and the two average based algorithms (average and mcquitty) and complete linkage tend to produce clusters of the same size (“circular clusters”)
- The two centroid based algorithms (centroid and median) might produce clusters of more varied sizes and the dendrogram show “inversions” that are difficult to interpret

```
# dev.new(width = 12/2.54, height = 22/2.54)

set.seed(12)
tmp <- matrix(runif(1000), ncol = 2)
colnames(tmp) <- c("x", "y")

k = 5
methods <- c("single", "complete", "average", "mcquitty", "centroid", "median",
             "ward.D", "ward.D2")

par(mfrow = c(4,2), mar = c(2,2,2,1), mgp = c(2, 0.6, 0), cex = 0.8, las = 1)
for(method in methods ){
  hcl <- hclust(dist(tmp[, c("x", "y")] ), method = method)
  groups <- cutree(hcl, k = k)
  plot(tmp[, c("x", "y")], col = groups, pch = groups, main = paste(method, "-", k, "groups"))
}
```

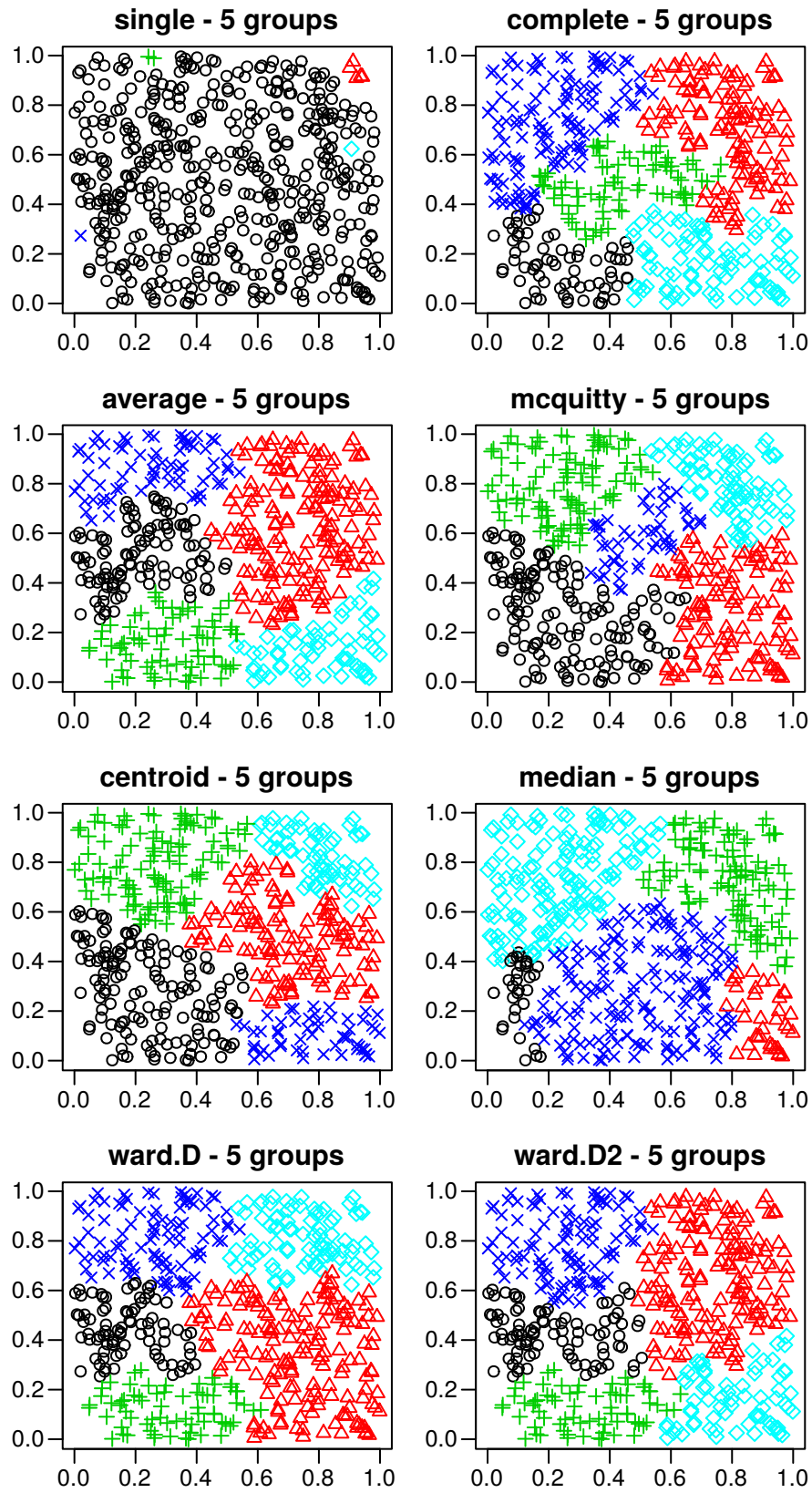


Figure 5:

The corresponding dendrograms.

```

# dev.new(width = 12/2.54, height = 22/2.54)
par(mfrow = c(4,2), mar = c(2,2,2,1), mgp = c(2, 0.6, 0), cex = 0.8, las = 1)
for(method in methods ){
  hcl <- hclust(dist(tmp[, c("x", "y")] ), method = method)
  plot(hcl, labels = FALSE, hang = -1, main = paste(method, "-", k, "groups"))
  rect.hclust(hcl, k = k)
}

```

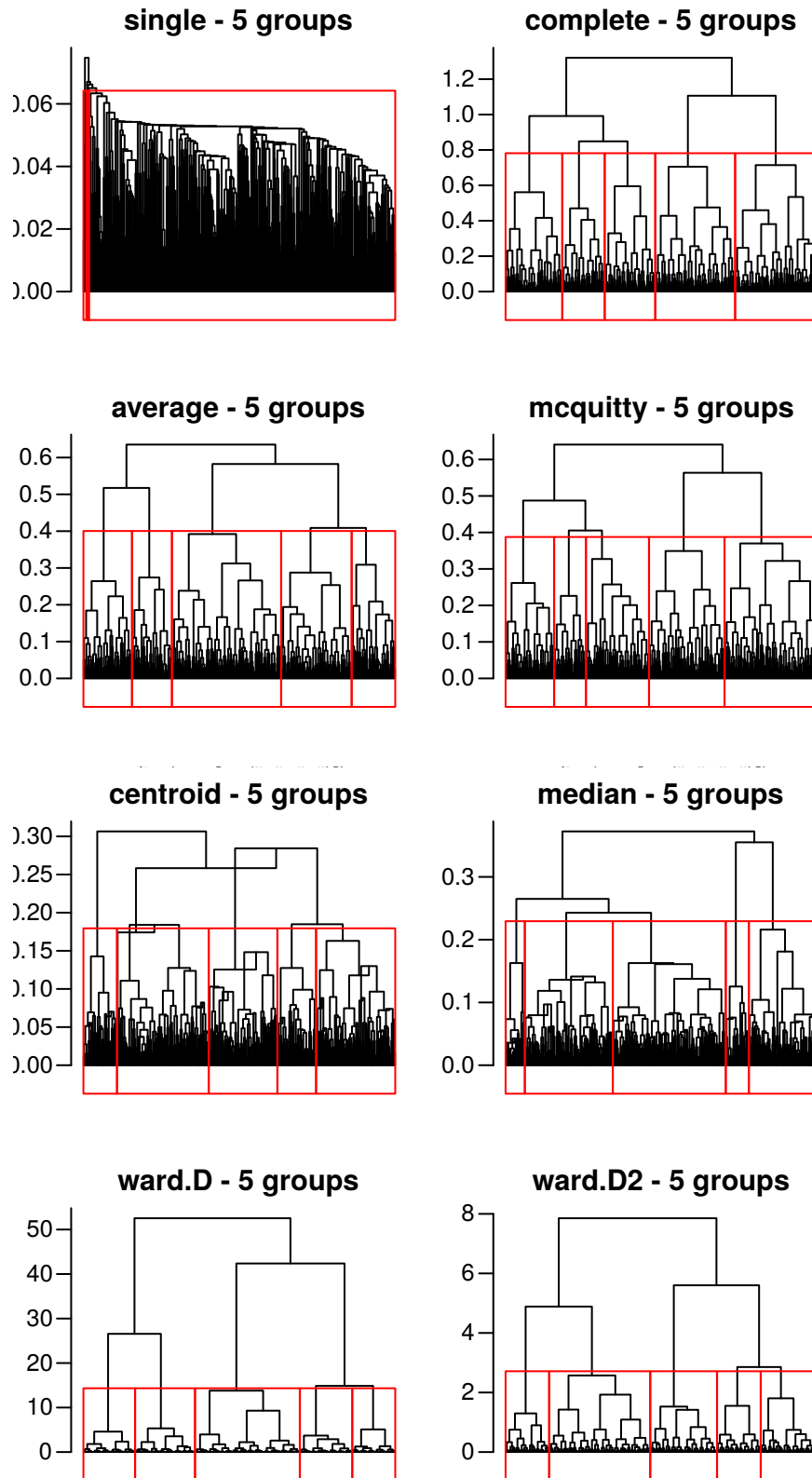


Figure 6:

We will now use a simple 2 dimensional simulated dataset to illustrate the difficulty to define a good clustering solution and how each method solves differently this question. Clusters are groups of similar observations. More specifically ideal clusters should have :

1. low within cluster distance : the distance between observations from the same cluster should be low
2. high between cluster distance : the distance between observations from different clusters should be high
3. discontinuity : you don't have intermediate observations that fall between clusters

These different criterion are not always compatible. In the following simulated example, the clusters have mainly been defined based on the discontinuity principle (there are “gaps” without observations between these 5 clusters). But the clusters have very different shapes and sizes (number of observations and within cluster distance). If you take into account the distances there are many incongruities in how these clusters have been defined. For example :

- Clusters 1 and 2 are very compact and clearly separated but their center are very close. The distance between observations from cluster 1 and 2 are lower than most of the distances between observations inside cluster 5. This cluster is much larger and contains many observations that are not very similar to each other.
- Cluster 4 is very elongated. The observations at the bottom of cluster 4 are closer to some observations in cluster 5 than to observations at the top of their own cluster
- Cluster 3 is rather different from any other cluster. Are cluster with only one observation really useful ???

```
set.seed(123)
cl1 <- cbind(x = rnorm(15, 3, 0.2), y = rnorm(15, 10, 0.2), cl = 1)
cl2 <- cbind(x = rnorm(15, 4.5, 0.2), y = rnorm(15, 10, 0.2), cl = 2)
cl3 <- cbind(x = 2, y = 7, cl = 3)
cl4 <- cbind(x = runif(20, 8, 8.5), y = runif(20, 6, 13), cl = 4)
cl5 <- cbind(x = runif(200, 0, 12), y = runif(200, 0, 5), cl = 5)
cl <- rbind(cl1, cl2, cl3, cl4, cl5)
```

Representation of the groups in 2 dimensions (here the full dimensions of this very simple dataset) :

```
# dev.new(width = 9/2.54, height = 7/2.54)
par(mar = c(2.5,2.5,2,4), mgp = c(1.7, 0.6, 0), cex = 0.8, las = 1)
plot(cl[, c("x", "y")], col = cl[, "cl"], pch = cl[, "cl"],
     main = '"True" clusters based on discontinuity')
legend("right", col = 1:5, pch = 1:5, legend = 1:5,
     xpd = NA, inset = -0.2, bty = "n", title = "Cluster")
```

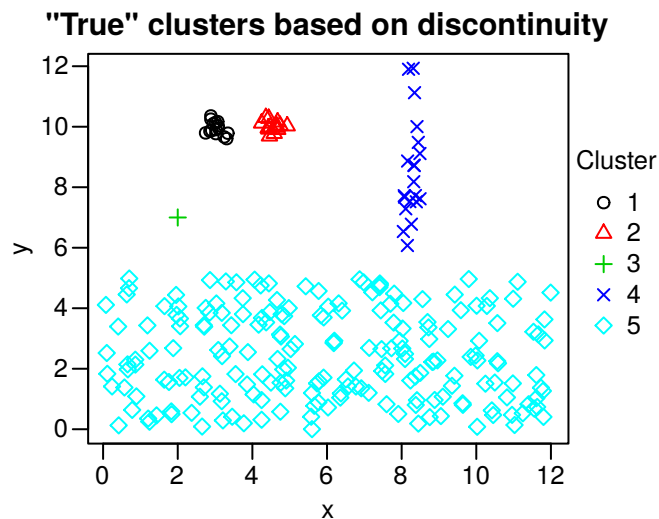


Figure 7:

Because hierarchical clustering is mainly based on distances, the solutions found by hclust are rather different. For example none of the methods consider that Cluster 1 and 2 are separated clusters. They will become separated clusters if you consider much more clusters and you will first need to divide cluster 5 in many smaller clusters.

The different grouping methods deal also differently with the outlier value forming cluster 3. For example the centroid methods tend to keep it in a separate cluster which might be useful if you need to spot unusual values. Other grouping methods like “ward” tend to add this outlier to one of the existing groups which might be useful if you want to ignore these unusual values and concentrate on the “main” patterns.

NB : other algorithms than Hierarchical Clustering can be based mainly on discontinuities (for example “DBSCAN”) and will find the same solution as the simulated one.

```
# dev.new(width = 12/2.54, height = 22/2.54)

par(mfrow = c(4,2), mar = c(2,2,2,1), mgp = c(2, 0.6, 0), cex = 0.8, las = 1)
for(method in methods ){
  hcl <- hclust(dist(cl[, c("x", "y")] ), method = method)
  groups <- cutree(hcl, k = k)
  plot(cl[, c("x", "y")], col = groups, pch = groups, main = paste(method, "-", k, "groups"))
}
```

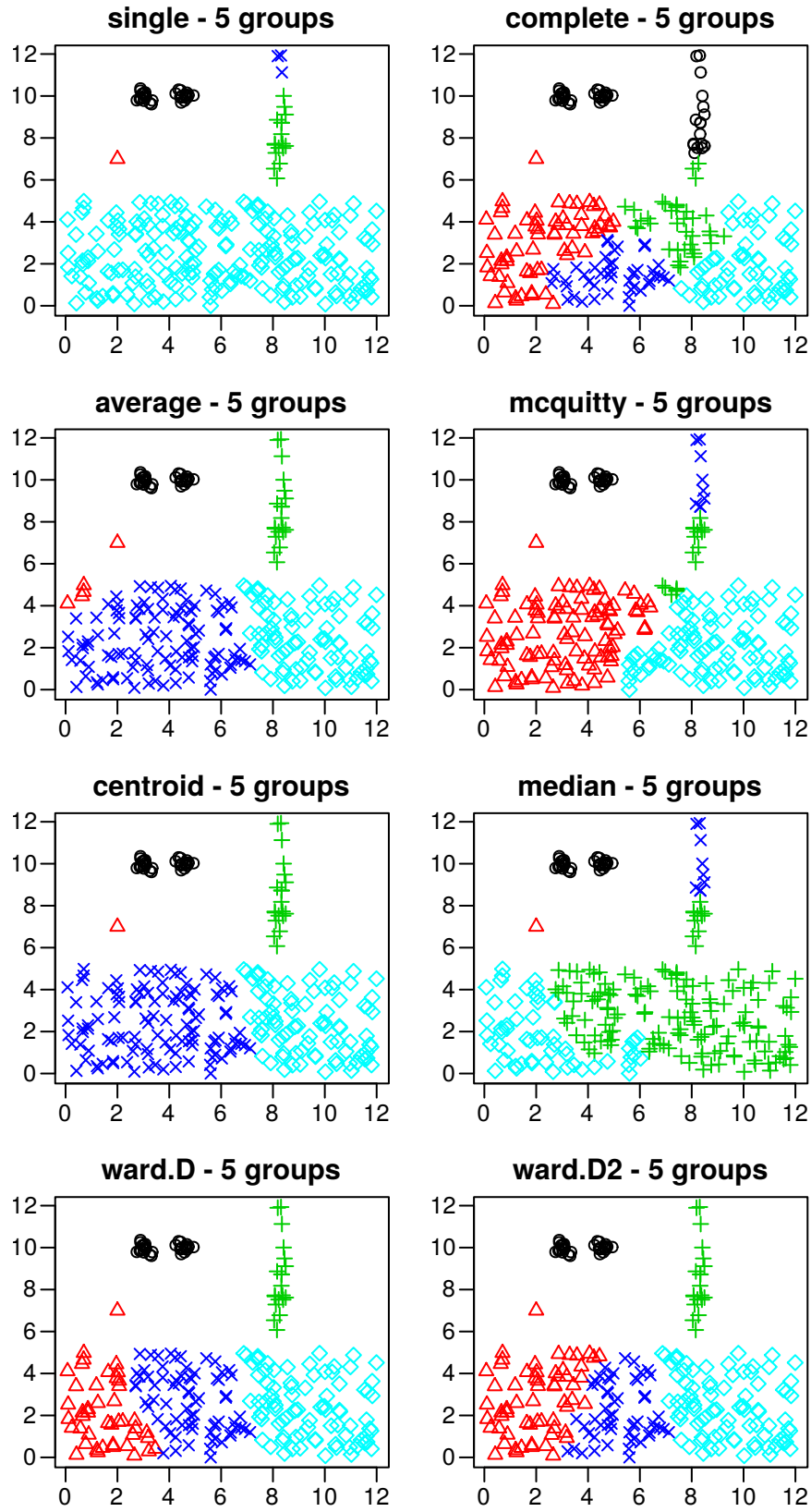


Figure 8:

The corresponding dendrograms. NB : the colors represent here the “true” clusters has they have been simulated. These colors do not correspond to the colors on the previous graphs

```

# dev.new(width = 12/2.54, height = 22/2.54)
par(mfrow = c(4,2), mar = c(2,2,2,1), mgp = c(2, 0.6, 0), cex = 0.8, las = 1)
for(method in methods ){
  hcl <- hclust(dist(cl[, c("x", "y")])), method = method)
  plot(hcl, labels = FALSE, hang = -1, main = paste(method, "-", k, "groups"))
  symbols(x = 1:nrow(cl), y = rep(-8*max(hcl$height)/100, nrow(cl)),
    rectangles= cbind(rep(1, nrow(cl)), max(hcl$height)/10),
    bg= cl[, "cl"][hcl$order], fg = NA, add=TRUE, inches=FALSE)
  rect.hclust(hcl, k = k)
}

```

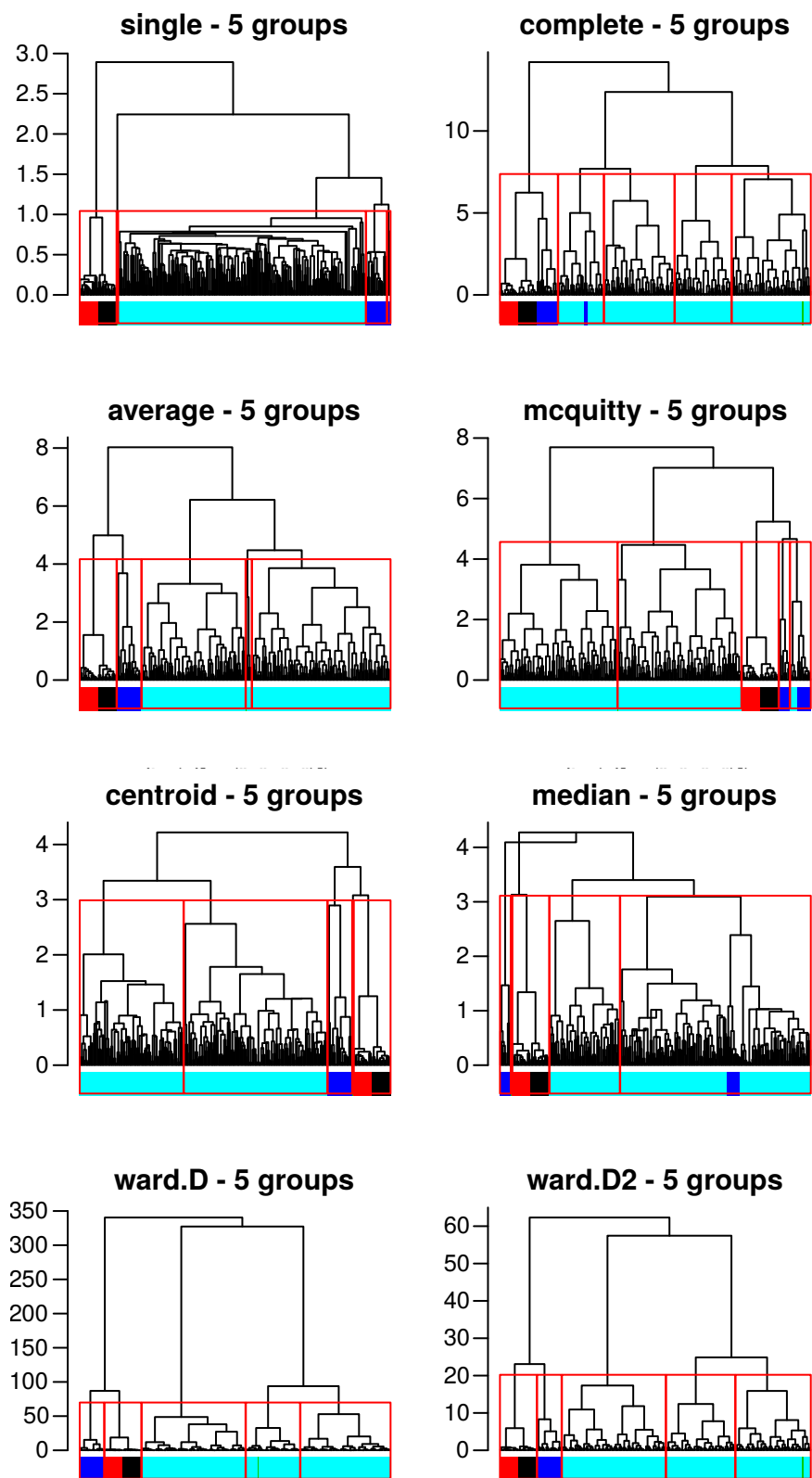


Figure 9:

1.3.4 Flexible-Beta clustering

It can be shown that all the grouping methods from `hclust` are particular cases of a “general agglomerative clustering model” described by 4 parameters : α_1 , α_2 , β and γ (see Legendre & Legendre 2012 p 367 and table 8.9).

Flexible-Beta clustering is a specific use of this generalized model that has gained a lot of popularity in Ecology in the recent years. β can take different values between -1 and +1 while the other parameters are fixed to be $\alpha_1 = \alpha_2 = (1-\beta)/2$ and $\gamma = 0$. Equivalently, $\beta = 1 - (\alpha_1 + \alpha_2)$. A β value close to 1 tends to produce long chaining (like single linkage) and negative values close to -1 tend to produce well separated groups. A value of $\beta = -0.25$ is very popular in community ecology and tends to produce intermediate results between average linkage and complete linkage.

This method is implemented in `cluster::agnes`

```
Eucl <- dist(cl[, c("x", "y")])
hcl <- cluster::agnes(Eucl, method='flexible',
                     par.method=c(0.625,0.625,-0.25, 0))
# by default, if par.method is of length 1 the value is the one of alpha_1 and the other
# parameters are constrained to produce the Flexible-Beta clustering approach.
# So you can obtain the same results as follows :
beta <- -0.25
hcl <- cluster::agnes(Eucl, method='flexible', par.method= (1-beta)/2)

# transform into hclust object and cut the tree...
hcl <- as.hclust(hcl)
groups <- cutree(hcl, k = k)

# Plot the results

# dev.new(width = 15/2.54, height = 7/2.54)
par(mfrow = c(1,2), mar = c(1.5,3.5,2.5,1), mgp = c(2, 0.6, 0), cex = 0.8, las = 1)
plot(hcl)
symbols(x = 1:nrow(cl), y = rep(-8*max(hcl$height)/100, nrow(cl)),
        rectangles= cbind(rep(1, nrow(cl)), max(hcl$height)/10),
        bg= cl[, "cl"][hcl$order], fg = NA, add=TRUE, inches=FALSE)
rect.hclust(hcl, k = k)

plot(cl[, c("x", "y")], col = groups, pch = groups, main = "Flexible Beta Clustering \n(Beta = -0.25)")
```

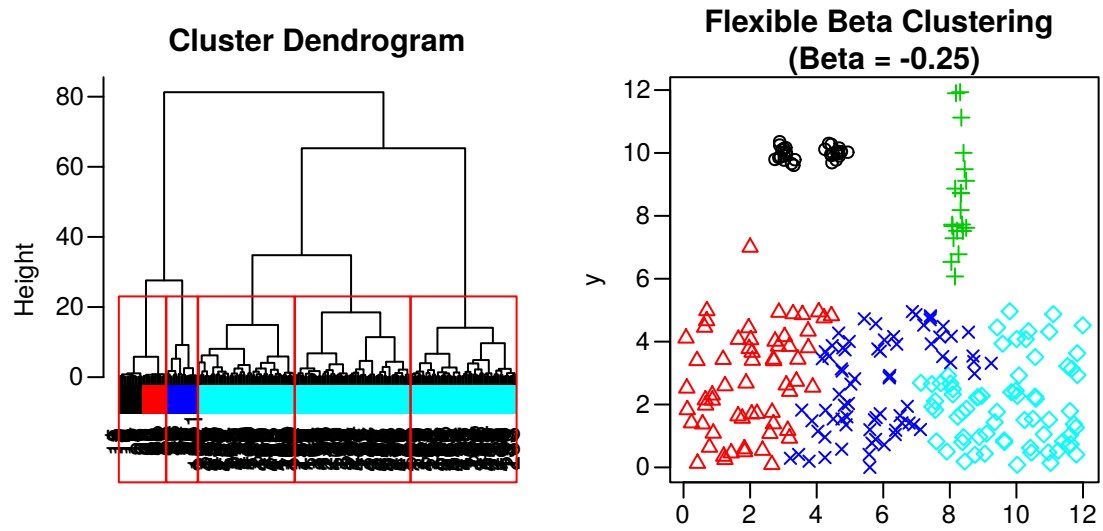


Figure 10:

1.3.5 Interpretation of the dendrogram and common pitfalls

The order of the labels is arbitrary. You should look only at the topology of the dendrogram to evaluate the distances and hierarchical relationships between the observations. This order can be changed (but remains constrained by the topology)

The length of the branches is not important. The position of the nodes represents the distance between observations or clusters.

2 dendrograms are stricly equivalent if their topology is the same (the branches lead to the same observations on the leaves) and the “height” of their nodes is the same.

For example the 3 following dendrograms are strictly equivalent (the order of the “leaves”, the length of the branches are diferent but the topology of the tree is identical and all the nodes are at the same position on the x axis).

The distance between A and B or F and G is 1. The maximum distance between C and the group AB is 4 (maximum distance because we used the default “complete linkage”).

In the two first dendrograms : G and D are next to each other on the x axis. However they are in completely different clusters and they are probably quite dissimilar. G is more similar to H than to D.

```
tmp <- c(1, 2, 5, 8, 10, 16, 17, 19)
names(tmp) <- LETTERS[seq_along(tmp)]
hcl <- hclust(dist(tmp))

# dev.new(width = 18/2.54, height = 6/2.54)

par(mfrow = c(1,3), mar = c(2,3.5,1,1), mgp = c(2, 0.6, 0), cex = 0.8, las = 1)
plot(hcl, main = "")
plot(hcl, hang = -1, main = "")
plot(reorder(hcl, wts = tmp), hang = -1, main = "")
```

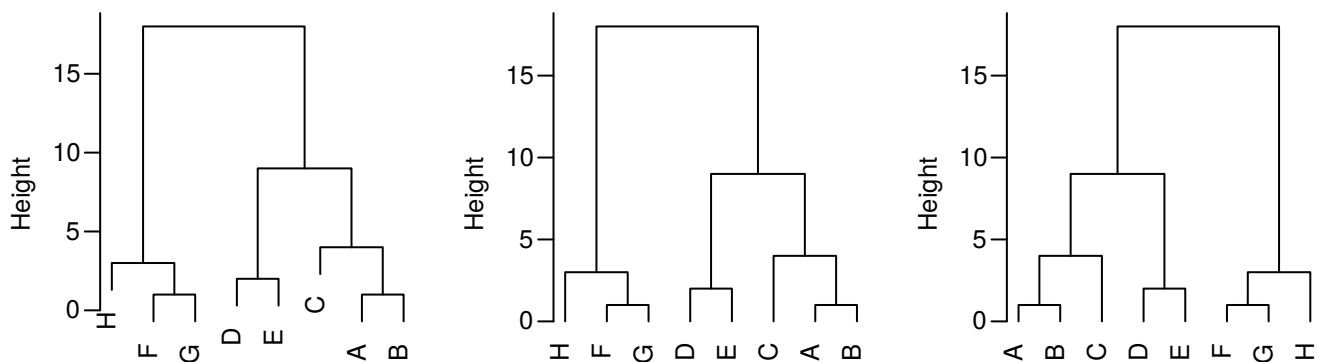


Figure 11:

1.3.6 Basic dendrograms manipulation and graphs

The typical operations you may want to do on a dendrogram :

- cut the tree at a given height or for a given number of groups to attribute each observation to a cluster (`cutree`)
- add information on the graph to visualize the groups (boxes, numbers, colors) (`rect.hclust`)
- add colors or symbols with external data (supplementary variable) to help the interpretation (`points`, `symbols`)
- zoom into the dendrogram or plot only a part of it when the dendrogram is too big
- reorder the leaves of the dendrogram (`reorder`)

The output of hierarchical clustering can be stored in two types (classes) of R objects : `hclust` (the default output of `hclust` function) and `dendrogram`. The `hclust` object is simpler and allows you to make basic plots that will cover 80% of your needs. `dendrogram` objects are more complex but they provide a much more diversified set of tools to manipulate and plot them (particularly with the addition of `dendextend` package). You can for example plot horizontal dendrograms, customize the branches appearance (color, line type), add a number on the branches for each group, ... You can always transform one type into the other with the functions `as.dendrogram` and `as.hclust`.

As an example dataset, we will use a subsample of the iris dataset.

```
set.seed(1)
d <- iris[sample(1:nrow(iris), 24),]
d[,1:4] <- scale(d[,1:4]) # standardise the first 4 columns
hcl <- hclust(dist(d[,1:4]), method = "ward.D")
```

The object contains many information that can be used eg the labels and the order of the observations in the dendrogram (relative to the original dataset)

```
str(hcl)

## List of 7
## $ merge      : int [1:23, 1:2] -13 -1 -22 -2 -14 -8 -4 -21 -5 -16 ...
## $ height     : num [1:23] 0 0.133 0.287 0.291 0.316 ...
## $ order      : int [1:24] 24 10 22 1 11 5 12 18 7 15 ...
## $ labels     : chr [1:24] "40" "56" "85" "134" ...
## $ method     : chr "ward.D"
## $ call       : language hclust(d = dist(d[, 1:4]), method = "ward.D")
## $ dist.method: chr "euclidean"
## - attr(*, "class")= chr "hclust"
```

You can cut the tree to classify the observations in different groups based on a desired number of groups (`k`) or the “height” on the dendrogram (`h`)

```
cutree(hcl, k = 3)

## 40 56 85 134 30 131 137 95 90 9 29 25 143 53 105 68 97 132 51 102 122 28 84
## 1 2 2 2 1 3 3 2 2 1 1 1 2 3 3 2 2 3 3 2 2 1 2
## 16
## 1
```

```
cutree(hcl, h = 5)

## 40 56 85 134 30 131 137 95 90 9 29 25 143 53 105 68 97 132 51 102 122 28 84
## 1 2 2 2 1 3 3 2 2 1 1 1 2 3 3 2 2 3 3 2 2 1 2
```

```
## 16
## 1
```

There is an equivalent function for dendrogram objects but only in the `dendextend` package. This function has more options. In particular you can chose to display the result in the order of the dataset (as for `hclust` objects) or in the order of the dendrogram. Note however that the numerotation is not the same (for example observation 40 can be in group “1” or “2”). The grouping solution is the same but the labels are different and arbitrary. This might have some consequences in the presentation of the results (see an example at the end of the section about heatmaps)

```
dend <- as.dendrogram(hcl)
dendextend::cutree(dend, k = 5, order_clusters_as_data = FALSE)
```

```
## 16  9  28  40  29  30  25 132 137 105 131  53  51  85  90  68  95  56  97 134  84 122 143
##  1  2  2  2  2  2  2  3  3  3  3  3  3  4  4  4  4  4  4  5  5  5  5
## 102
##  5
```

```
dendextend::cutree(dend, k = 5, order_clusters_as_data = TRUE)
```

```
## 40  56  85 134  30 131 137  95  90  9  29  25 143  53 105  68  97 132  51 102 122  28  84
##  1  2  2  3  1  4  4  2  2  1  1  1  3  4  4  2  2  4  4  3  3  1  3
## 16
##  5
```

To obtain a clustering numerotation with numbers matching the cluster order (here observation 40 is well in group 2):

```
tmp <- dendextend::cutree(dend, k = 5, order_clusters_as_data = FALSE)
tmp[order(order.dendrogram(dend))]
```

```
## 40  56  85 134  30 131 137  95  90  9  29  25 143  53 105  68  97 132  51 102 122  28  84
##  2  4  4  5  2  3  3  4  4  2  2  2  5  3  3  4  4  3  3  5  5  2  5
## 16
##  1
```

Plotting and adding boxes to show the groups is easy. It is possible to add annotations on the groups below or above the boxes but this is not very straightforward...

NB : the colors here are not very useful...

```
# dev.new(width = 12/2.54, height = 8/2.54)
par(mar = c(3.5,3.5,1,1), mgp = c(2, 0.6, 0), cex = 0.8)
plot(hcl, hang = -1)
rect.hclust(hcl, k = 3, border = c("dodgerblue", "orangered", "gold"))

# compute the position of the labels on the x axis (this is the crazy part ;- )
cluster <- cutree(hcl, k = 3)
clustab <- table(cluster)[unique(cluster[hcl$order])]
xposition <- cumsum(c(0.5,clustab[-length(clustab)])) + clustab/2

# add this text in the bottom margin
mtext(text = c("Group 1", "Group 2", "Group 3"),
      side = 1, line = 1, at = xposition, adj = 0.5,
      col = c("dodgerblue", "orangered", "gold"))
# add the same text on top of the boxes
```

```
text(labels = c("Group 1", "Group 2", "Group 3"),
     x = xposition, y = 8, adj = 0.5,
     col = c("dodgerblue", "orangered", "gold"))
```

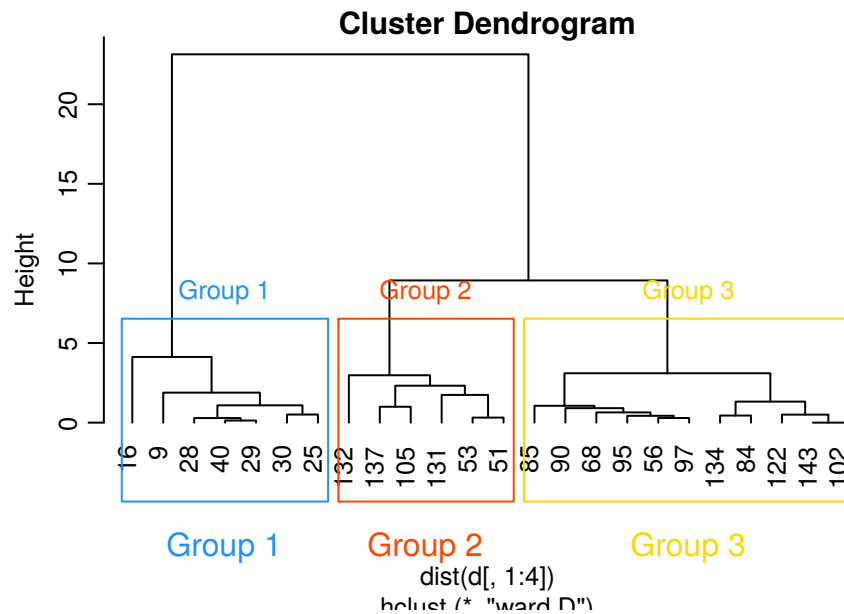


Figure 12:

Add information from external data : here the species. Change the labels, color the labels and add colored symbols for each species. Note that you must respect the order of the observations in the dendrogram provided by `hcl$order`.

```
# plot without labels, x axis label (`xlab`), title (`main`) and subtitle (`sub`)
# dev.new(width = 12/2.54, height = 8/2.54)
par(mar = c(3.5,3.5,1,1), mgp = c(2, 0.6, 0), cex = 0.8)
plot(hcl, hang = -1, labels = FALSE, sub = "", xlab = "", main = "")
# add manually the labels (`las=2` to have vertical labels, `cex=0.8` for smaller labels)
mtext(text = d$Species[hcl$order], side = 1, line = -0.5, at = 1:nrow(d),
      las = 2, cex = 0.8,
      col = c("dodgerblue", "orangered", "gold")[d$Species[hcl$order]])
# add symbols
points(x = 1:nrow(d), y = rep(-0.5, nrow(d)), cex = 1.5,
      pch = c(15, 16, 17)[d$Species[hcl$order]], # chape of the points
      col = c("dodgerblue", "orangered", "gold")[d$Species[hcl$order]])
# add legend
legend(x = "topright", pch = c(15, 16, 17), legend = levels(d$Species),
      col = c("dodgerblue", "orangered", "gold"), bty = "n", pt.cex = 1.5)
```

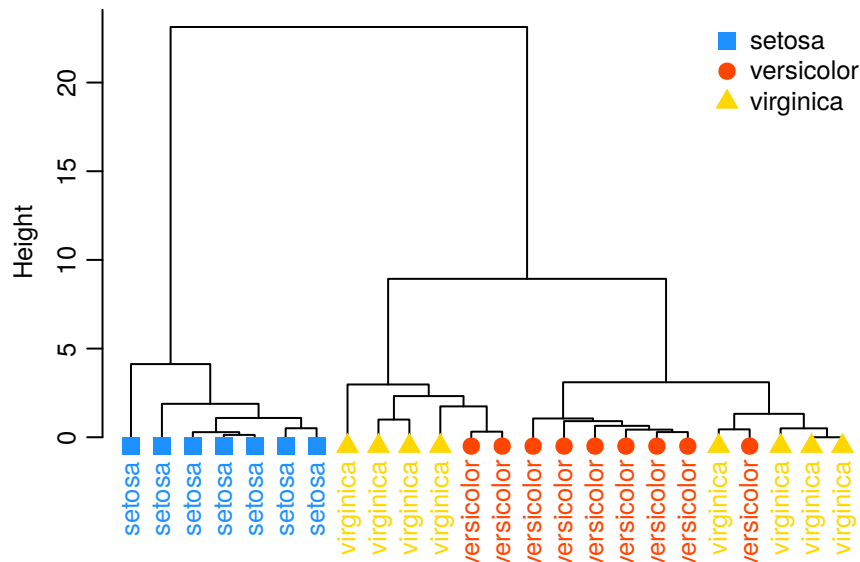


Figure 13:

When there are many observations, adding labels or symbols is not an option because the graph will quickly become difficult to read. In that case adding a colored band is generally the best option to visualize an external supplementary variable. This can be done with the `symbols` function.

```
# dev.new(width = 8/2.54, height = 6/2.54)
par(mar = c(1.5,3.5,1,1), mgp = c(2, 0.6, 0), cex = 0.8)
plot(hcl, labels = FALSE, hang = -1, main = "")
symbols(x = 1:nrow(d), y = rep(-2, nrow(d)),
        rectangles= cbind(rep(1, nrow(d)), 3),
        bg= c("dodgerblue", "orangered", "gold")[d$Species[hcl$order]],
        fg = NA, add=TRUE, inches=FALSE)
legend(x = "topright", legend = levels(d$Species), border = NA,
       fill = c("dodgerblue", "orangered", "gold"), bty = "n")
```

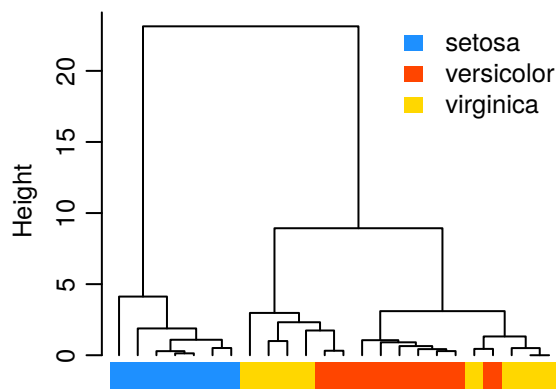


Figure 14:

The following code chunk is a little more complicated but it will be independent of the y axis scale. (the result is identical and not shown here)

```
plot(hcl, labels = FALSE, hang = -1, main = "")
symbols(x = 1:nrow(d), y = rep(-8*max(hcl$height)/100, nrow(d)),
```

```

rectangles= cbind(rep(1, nrow(d)), max(hcl$height)/10),
bg= c("dodgerblue", "orangered", "gold")[d$Species[hcl$order]],
fg = NA, add=TRUE, inches=FALSE)
legend(x = "topright", legend = levels(d$Species), border = NA,
fill = c("dodgerblue", "orangered", "gold"), bty = "n")

```

You can reorder an hclust object with a dedicated function from package **vegan**. Here we reorder the observations along the first axis of a principal component analysis. this is not very usefull here but in many circumstances this can improve the readability of the dendrogram. (NB dendrogram objects can be reordered directly with the `reorder.dendrogram` function)

```

# dev.new(width = 8/2.54, height = 6/2.54)
par(mar = c(2.5,3.5,1,1), mgp = c(2, 0.6, 0), cex = 0.8)

pca <- princomp(d[, -5])
tmp <- reorder(hcl, wts = order(pca$scores[, "Comp.1"]))
plot(tmp, hang = -1, labels = FALSE, sub = "", xlab = "", main = "")
# add manually the labels (`las=2` to have vertical labels, `cex=0.8` for smaller labels)
mtext(text = d$Species[tmp$order], side = 1, line = -0.5, at = 1:nrow(d),
las = 2, cex = 0.6,
col = c("dodgerblue", "orangered", "gold")[d$Species[tmp$order]])

```

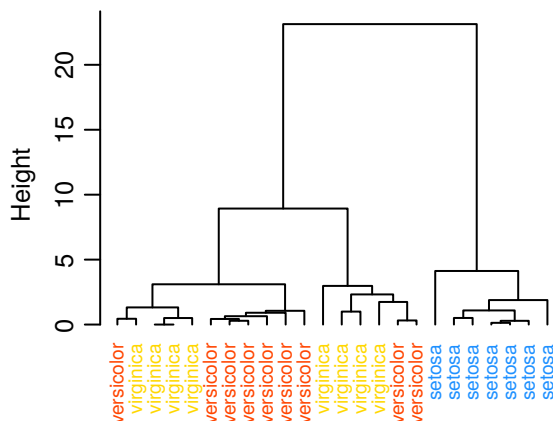


Figure 15:

hclust objects are simpler and will do the job most of the time. With **dendrogram** object you can do much more. See the **dendextend** vignette to have an idea of the possibilities. However keep in mind that it is often better to keep the graphs as simple as possible and that colors are often misused in graphical representation.

Here are a few things that you can do with **dendrogram** objects and that are impossible or very difficult to do with **hclust** objects.

Plot a horizontal dendrogram :

```

# dev.new(width = 10/2.54, height = 8/2.54)

dend <- as.dendrogram(hcl)
par(mar = c(2.5,0.5,0.5,2), mgp = c(2, 0.6, 0), cex = 0.7)
plot(dend, horiz = TRUE)

```

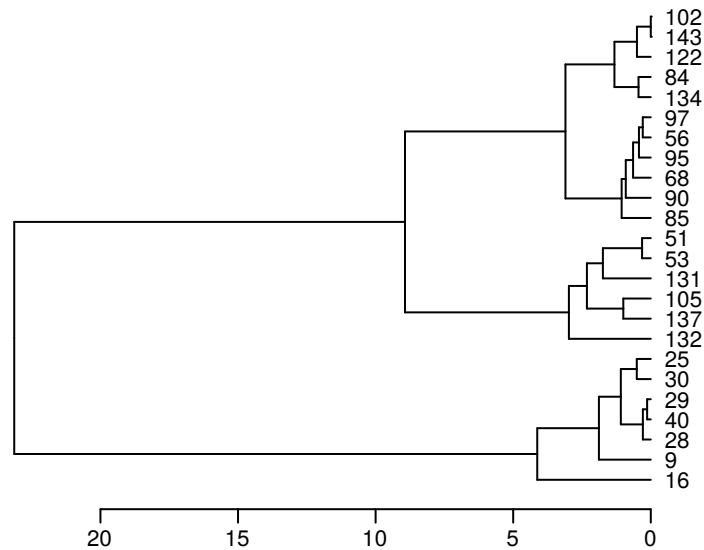


Figure 16:

Zoom into the dendrogram to plot only one of the groups with `xlim` and `ylim`. Very useful when the dendrogram is too big and you want to examine more closely one of the clusters... However this is not perfect because as you can see, labels from another cluster are still visible below the x axis

```
# dev.new(width = 8/2.54, height = 6/2.54)
par(mar = c(2.5,0.5,0.5,2), mgp = c(2, 0.6, 0), cex = 0.7)
plot(dend, horiz = TRUE, main = "", ylim = c(13.5, 24.5), xlim = c(5,0))
# add symbols
points(y = 14:24, x = rep(0, nrow(d))[14:24], cex = 1.5,
       pch = c(15, 16, 17)[d$Species[hcl$order]][14:24], # shape of the points
       col = c("dodgerblue", "orangered", "gold")[d$Species[hcl$order]][14:24])
# add legend
legend(x = "topleft", pch = c(15, 16, 17), legend = levels(d$Species),
       col = c("dodgerblue", "orangered", "gold"), bty = "n", pt.cex = 1.5)
```

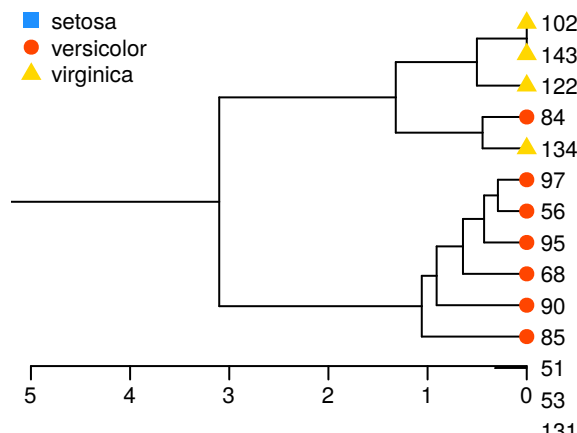


Figure 17:

An other and arguably better solution is to use the dedicated function from `dendextend` that will create a list with the “sub-dendrograms”

```
# dev.new(width = 8/2.54, height = 6/2.54)
library(dendextend)
dend_list <- get_subdendrograms(dend, k = 3)
par(mar = c(2.5,0.5,0.5,2), mgp = c(2, 0.6, 0), cex = 0.7)
plot(dend_list[[2]], horiz = TRUE)
points(y = 1:11, x = rep(0, nrow(d))[14:24], cex = 1.5,
       pch = c(15, 16, 17)[d$Species[hcl$order]][14:24], # shape of the points
       col = c("dodgerblue", "orangered", "gold")[d$Species[hcl$order]][14:24])
# add legend
legend(x = "topleft", pch = c(15, 16, 17), legend = levels(d$Species),
      col = c("dodgerblue", "orangered", "gold"), bty = "n", pt.cex = 1.5)
```

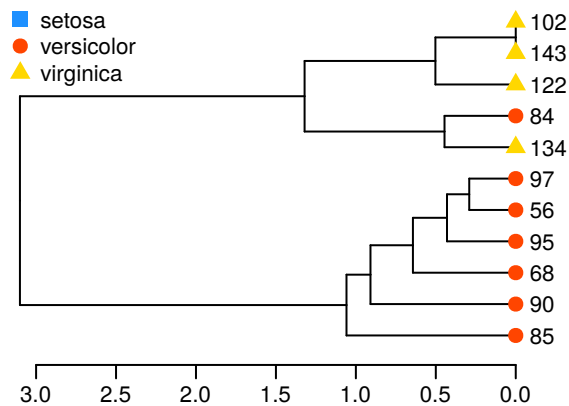


Figure 18:

You can also cut the tree at a certain height (k groups option is not available). You then obtain a list of trees above or below the cut

```
tmp <- cut(dend, h = 5 )
tmp
```

```
## $upper
## 'dendrogram' with 2 branches and 3 members total, at height 23.13304
##
## $lower
## $lower[[1]]
## 'dendrogram' with 2 branches and 7 members total, at height 4.125621
##
## $lower[[2]]
## 'dendrogram' with 2 branches and 6 members total, at height 2.975649
##
## $lower[[3]]
## 'dendrogram' with 2 branches and 11 members total, at height 3.101122
```

```
# dev.new(width = 8/2.54, height = 5/2.54)
par(mar = c(2.5,0.5,0.5,2), mgp = c(2, 0.6, 0), cex = 0.7)
plot(tmp$upper, horiz = TRUE)
```

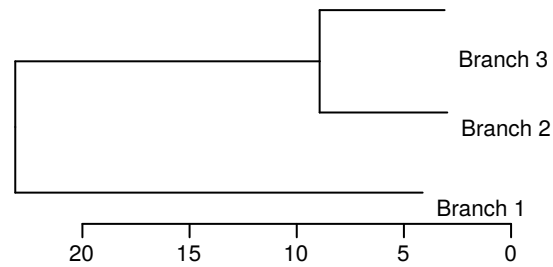


Figure 19:

```
plot(tmp$lower[[3]], horiz = TRUE)
```

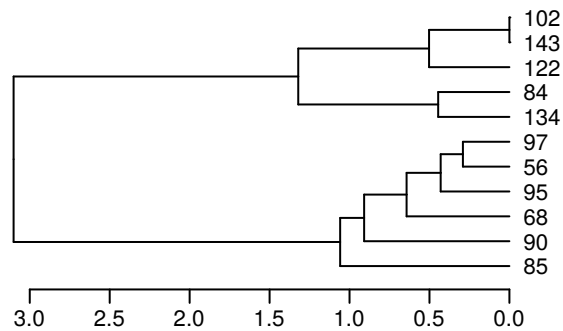


Figure 20:

Easier way to color the labels and add a number to the clusters. Remember that the group numbers provided here will generally be different from the group numbers provided by `cutree` (based on the order of the dataset)

```
# dev.new(width = 10/2.54, height = 8/2.54)
par(mar = c(2.5,0.5,0.5,2), mgp = c(2, 0.6, 0), cex = 0.7)
tmp <- dend
labels_colors(tmp) <- c("dodgerblue", "orangered", "gold")[d$Species[order.dendrogram(tmp)]]
tmp <- color_branches(tmp, col = "black", groupLabels=TRUE, k = 3)
plot(tmp, horiz = TRUE)
rect.dendrogram(tmp, k = 3, horiz = TRUE)
```

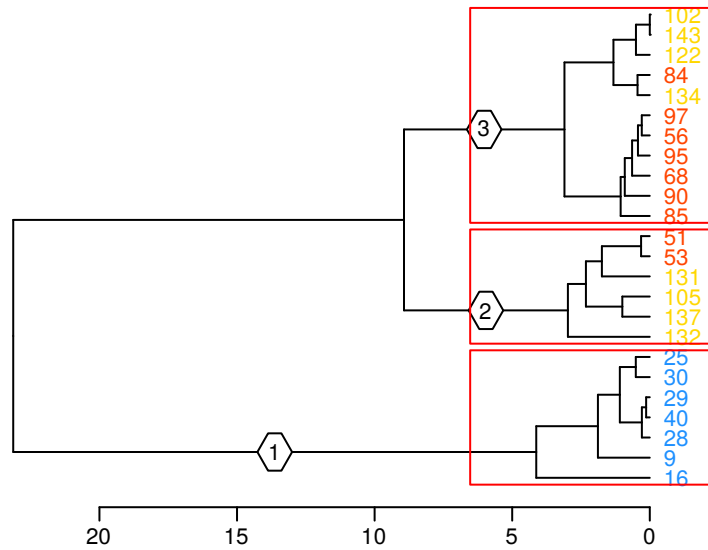



Figure 21:

It is also possible to add cluster labels at different levels. Here for example Group A contains groups 1 and 2

```
# dev.new(width = 10/2.54, height = 8/2.54)
par(mar = c(2.5,0.5,0.5,2), mgp = c(2, 0.6, 0), cex = 0.7)
tmp <- dend
labels_colors(tmp) <- c("dodgerblue", "orangered", "gold")[d$Species[order.dendrogram(tmp)]]
tmp <- color_branches(tmp, col = "black", groupLabels=c("A", "B"), k = 2)
tmp <- color_branches(tmp, col = "black", groupLabels=TRUE, k = 4)
plot(tmp, horiz = TRUE)
```

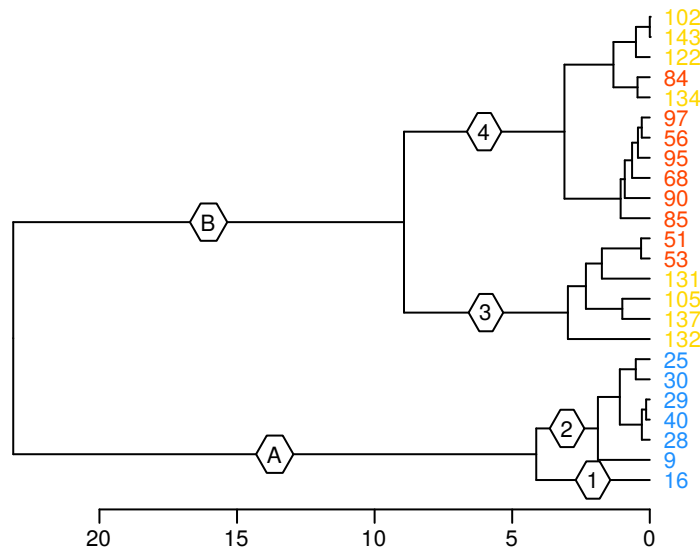


Figure 22:

1.4 K-means and K-medoids partitioning

K-means and K-medoids are two related non hierarchical clustering approaches. The user have to decide the number of clusters wanted and the algorithms will create groups that minimize the within group (squared) Euclidean distances (k-means) or any distance index (k-medoids). If a Hellinger tranformation is applied on the dataset before applying the k-means algorithms, the distance minimized is the Hellinger distances which makes the k-means algorithm adapted for species composition data (minimizing the effect of double 0).

A widely used algorithm for k-medoids clustering is PAM = Partitioning Around Medoids. PAM and k-medoids are often used as synonyms.

Both algorithms find their solution by an iterative process. As a consequence you might end up with slightly different results if you repeat the analysis several time. A common practice is to run the analysis several time and take an average of the solutions or a majority vote.

There are some differences between the algorithms :

- k-means works on the raw data (and computes Euclidean distances) while PAM works on a distance matrix
- k-means is based on centroids ie the average of the variables of each cluster that do not correspond to a really existing observation/point. PAM is based on medoids ie really existing observations/points that are considered as characteristic of the cluster

Example with the scaled iris dataset and the Euclidan distance for both methods :

```
iris_scaled <- scale(iris[,1:4])
iris_euclid <- dist(iris_scaled)

set.seed(123) # to be sur to obtain the same result at each time
# : nstart = 50 --> the k-means is repeated 50 times and a consensus solution is provided
KM <- kmeans(iris_scaled, centers = 3, nstart = 50)
PAM <- cluster::pam(iris_euclid, k = 3)
```

Clusters and centroids of the k-means

[illegible]

Clusters and medoids of the PAM

```
PAM$clustering  
## [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
## [46] 1 1 1 1 1 2 2 2 3 3 3 2 3 3 3 3 3 3 3 3 2 3 3 3 3 3 3 3 3 2 2 2 3 3 3 3 3 3 3 3 3 2 3 3 3 3  
## [91] 3 3 3 3 3 3 3 3 3 3 2 3 2 2 2 2 3 2 2 2 2 2 2 3 2 2 2 2 3 2 3 2 3 2 2 3 3 2 2 2 2 2 3 3  
## [136] 2 2 2 3 2 2 2 3 2 2 2 3 2 2 3
```

```
iris_scaled[PAM$medoids,]
```

```
##      Sepal.Length Sepal.Width Petal.Length Petal.Width
## [1,]   -1.0184372    0.7861738   -1.2791040   -1.3110521
## [2,]    1.1553023   -0.1315388    0.9868021    1.1816087
## [3,]   -0.1730941   -0.5903951    0.4203256    0.1320673
```

compare the clustering solution of the 2 methods : only 4 differences /150

```
table(KM$cluster, PAM$clustering)
```

```
##
##      1  2  3
## 1 50  0  0
## 2  0 44  3
## 3  0  1 52
```

To have an idea of the best number of clusters for PAM you can compute it for several values of k and compare the silhouette width (the higher the better). See also the section about cluster validation...

```
maxk <- 15
PAM <- as.list(vector(length = maxk))
sil_width <- vector(length = maxk)
for(k in 2:15) {
  PAM[[k]] <- cluster::pam(iris_euclid, k = k)
  sil_width[k] <- PAM[[k]]$silinfo$avg.width
}
```

```
# dev.new(width = 10/2.54, height = 7/2.54)
par(mar = c(3.5,3.5,1,1), mgp = c(2, 0.6, 0), cex = 0.8, las = 1)
plot(sil_width, type = "b", xlab = "Number of Clusters")
abline(v = which.max(sil_width), lty = 2, col = "gray50")
```

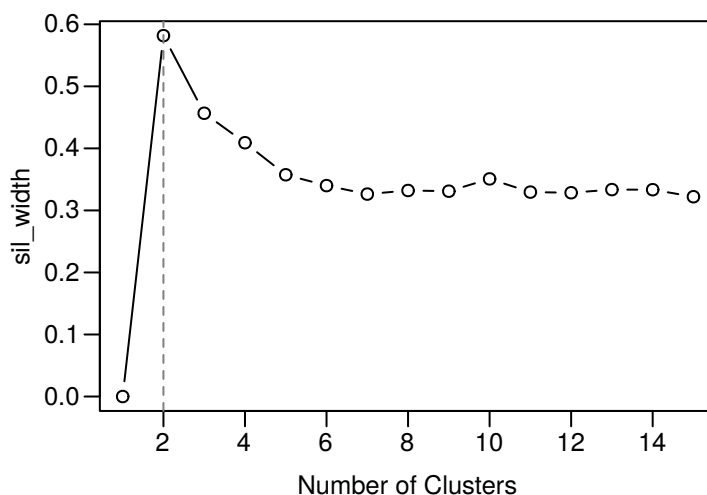


Figure 23:

For k-means, **vegan** provides a nice `cascadeKM` function that automatize the whole process. It provides a graphical method and computes the “calinski” criterion (or the ssi criterion) to help you choose the number of clusters. Here the best - the maximum in red - is = 2. According to the help of the function

: " Points marked in orange, if any, indicate partitions producing an increase in the criterion value as the number of groups increases; they may represent other interesting partitions."

See also the section about cluster validation...

```
KM <- vegan::cascadeKM(iris_scaled, 2, 15)
```

```
# dev.new(width = 16/2.54, height = 10/2.54)  
plot(KM)
```

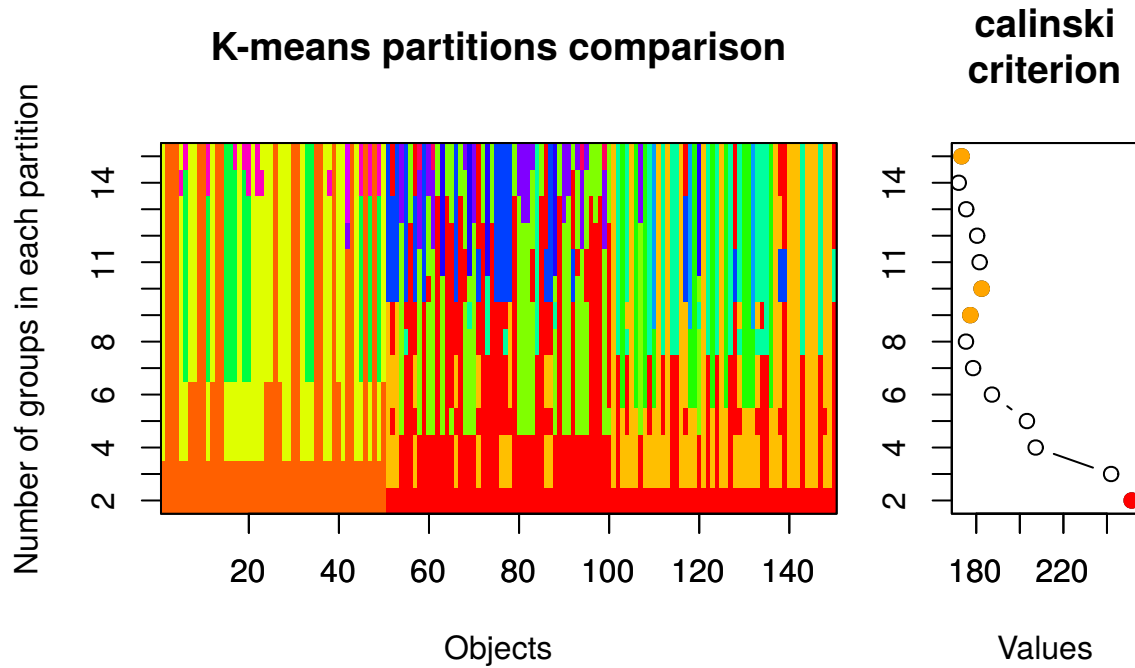


Figure 24:

1.5 Clusters interpretation

For this section, we will work on a real dataset : pollen pellets collected by honey bees have been collected in several sites in July, August, September and October. ~ 1000 pollen grains have been extracted and identified up to the family, genus or species level.

```
d <- read.csv2("data/pollen/full_dataset.csv")
# d <- d[d$Buffer == 1500, ]
d <- data.frame(d[, c(2:5)], d[, 14:47])
d <- unique(d)
d <- na.omit(d)
d <- d[,c(1:4, which(colSums(d[, -c(1:4)]) > 50) + 4)]

# Always check your data import !!
# dim(d)
# summary(d)
```

1.5.1 Clusters visualisation

1.5.1.1 Visualize the clusters with heatmaps

The main advantage of heatmaps (for hierarchical clustering) is that you can visualize the whole dendrogram in front of the data matrix without choosing in advance the number of clusters you will consider. This can really help you to decide which clusters are interpretable and how many clusters you want to use. All other methods require that you “cut” first your dendrogram to define the groups.

```
# select the columns to use
Y <- d[, -c(1:4)]
# Y <- Y[, -which(colnames(Y) == "cen")]

# add a row name composed of the Apiary code and the number of the month
d$Month <- factor(d$Month, levels = c("July", "August", "September", "October"))
rownames(Y) <- paste0(d$IDApiary, "_",
                      sprintf("%02.0f", as.numeric(d$Month) + 6))
```

Simple heatmaps with default clustering (euclidean distance and complete linkage -> not very well adapted here)

```
# dev.new(width = 18/2.54, height = 14/2.54)
gplots::heatmap.2(as.matrix(Y))
```

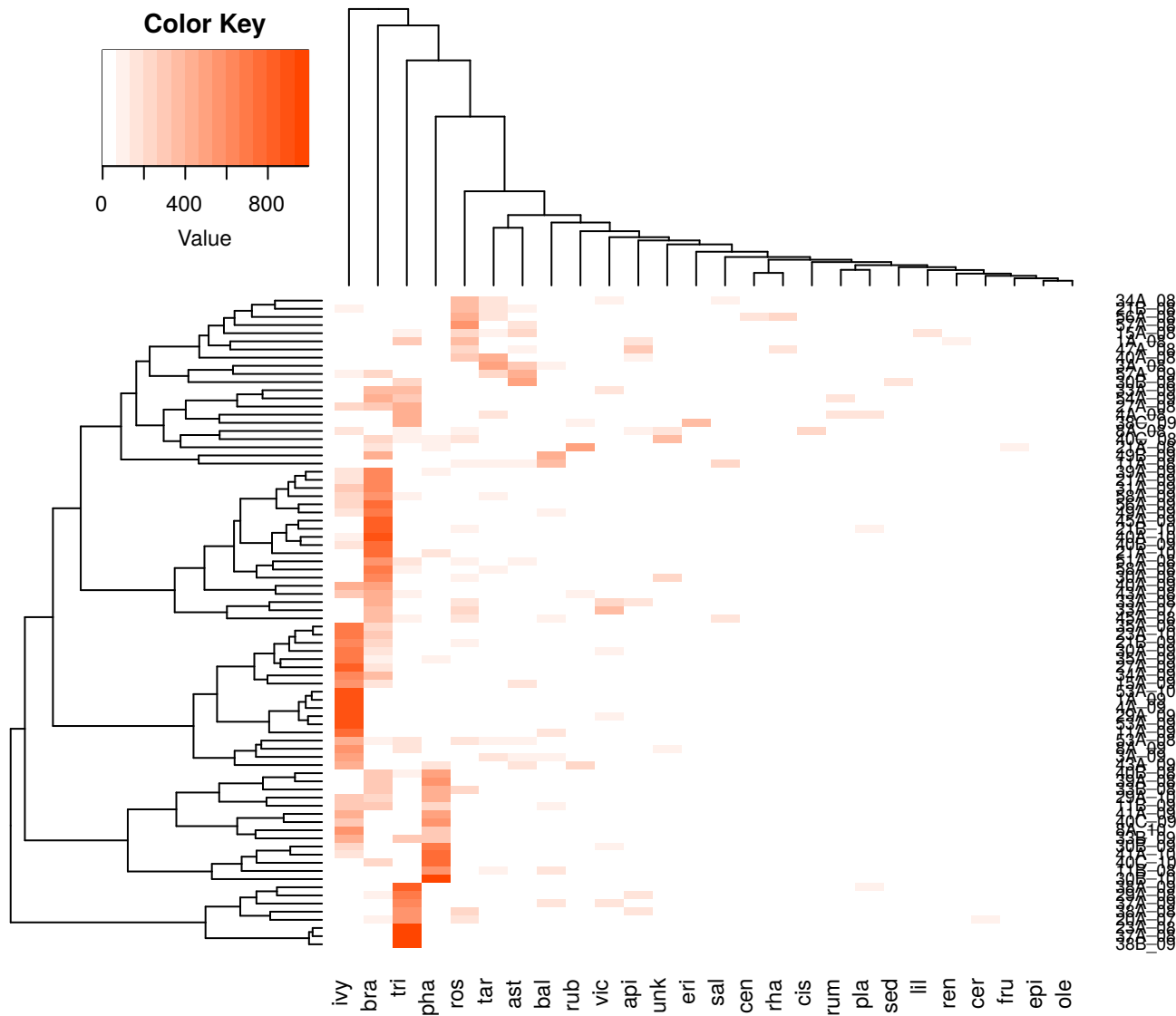



Figure 26:

Custom dendrogram based on Hellinger distance.

```
# transformation and compute the distances
hell_trans <- decostand(Y, "hellinger") # hellinger transformation
hell_dist <- dist(hell_trans) # hellinger distance between the rows (samples)
# correlation between the columns (plant species) on the hellinger transformed data
# Note that this must be later transformed into a distance measure with `as.dist(1-R)`
hell_cor <- cor(hell_trans)

rowClust <- hclust(hell_dist, method = "ward.D2") # ward clustering on the rows
colClust <- hclust(as.dist(1-hell_cor), method = "ward.D2") # clustering on the columns

# dev.new(width = 18/2.54, height = 16/2.54)
heatmap.2(as.matrix(Y),
  Colv = as.dendrogram(colClust), # column dendrogram with a dendrogram class
  Rowv = as.dendrogram(rowClust),
  col = colorRampPalette(c("white", "orangered")),
```

```
trace = "none", density.info=c("none"))
```

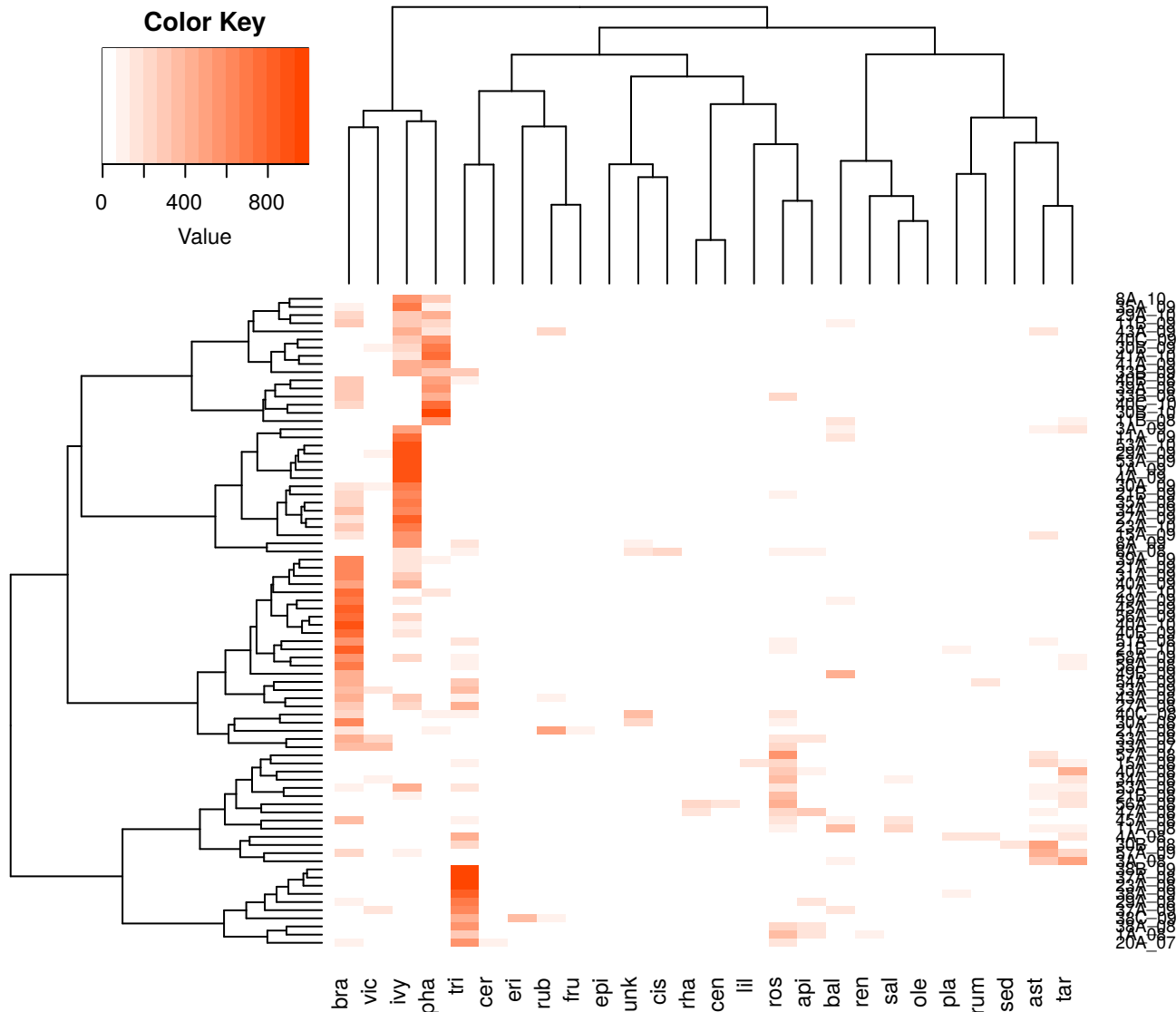


Figure 27:

Highly customized heatmap...

The code is much longer but in my opinion this is really worth the effort. You will end up with a graphical representation of your data that is highly complex but nevertheless readable. It is difficult to create a function that will automate this because each case is slightly different and you have to make different choices for each dataset. It is however quite easy to copy-paste that code and adapt it to your needs.

Here the color palette is a discrete gradient. When you have variables with different units it will not be possible to use one unique gradient for all of them. In this situation the best option is generally to standardise the data (for each column remove the mean and divide by the standard deviation) and then represent in one color the values that are much below average (eg $< \text{average} - 1 * \text{SD}$) and in another color the values that are much higher than average (eg $> \text{average} + 1 * \text{SD}$).

```
# custom dendrograms
# transformation and compute the distances
hell_trans <- decostand(Y, "hellinger") # hellinger transformation
```



```

hell_dist <- dist(hell_trans) # hellinger distance between the rows (samples)
# correlation between the columns (plant species) on the hellinger transformed data
# Note that this must be later transformed into a distance measure with `as.dist(1-hell_cor)`
hell_cor <- cor(hell_trans)

rowClust <- hclust(hell_dist, method = "ward.D2") # ward clustering on the rows
colClust <- hclust(as.dist(1-hell_cor), method = "ward.D2") # clustering on the columns

# reorder the dendrograms
# This is optional but it generally improves readability.
# However, you must be extra-careful when you need
# to find positions in the dendrogram after the reordering
rowClust <- reorder(rowClust, rowMeans(Y))
colClust <- reorder(colClust, rev(colMeans((1-hell_cor))))

# Optional but recommended : create a color palette with custom breaks
# We want white when the pollen has not been observed (= 0) then a gradient from
# yellow to red
mybreaks <- c(-1,1, 100, 250, 500, 750, 1000)
mypalette <- c("white",
               colorRampPalette(c("lightyellow", "gold", "red2"))(length(mybreaks)-2))

# Optional : Store a temporary matrix of the data with the 0 replaced by NA
# The values of this matrix will be displayed on the heatmap
values <- Y
values[values==0] <- NA

# Optional : Supplementary variable : different colors for each month
# you might also for example add a scale for the silhouette statistic to visualise
# which observations are probably missclassified or between 2 clusters
grcol <- c("lightgreen", "forestgreen", "lightblue", "steelblue")
# grcol <- brewer.pal(4, "Paired")[c(3, 4, 1, 2)]
grcol <- grcol[d$Month]

# Optional : find positions to add horizontal and vertical bars to separate the groups.
# Here we choose arbitrarily 3 groups for the species (columns) and 5 groups for the
# rows (samples)
tmp <- cutree(colClust, k = 3)[colClust$order]
colsep <- which(tmp[-1]!=tmp[-length(tmp)]) # where do we change the group ?
tmp <- cutree(rowClust, k = 5)[rev(rowClust$order)]
rowsep <- which(tmp[-1]!=tmp[-length(tmp)])

# Optional : add a number and color the branches of the row dendrogram according
# to the groups
library(dendextend)
branchcol = c("forestgreen", "lightgreen", "dodgerblue", "orange", "purple")
rowClust_withGroups <- as.dendrogram(rowClust)
rowClust_withGroups <- color_branches(rowClust_withGroups, col = branchcol,

```

```
groupLabels=TRUE, k = 5)
```

```
# dev.new(width = 18/2.54, height = 22/2.54)
heatmap.2(as.matrix(Y), # must be a numeric matrix not a data.frame
  trace = "none", density.info=c("none"),

  Colv = as.dendrogram(colClust), # column dendrogram with a dendrogram class
  Rowv = rowClust_withGroups,
  breaks = mybreaks,
  col = mypalette,

  cellnote = values, notecex = 0.8, notecol="gray25", # numbers in the cells

  RowSideColors = grcol, # supplementary variable (here monthes)

  # block separation (horizontal and vertical lines)
  colsep = colsep, rowsep = rowsep, sepcolor="black",

  # size of the text and different parts of the graph
  cexRow = 0.85, cexCol = 1.25, offsetCol = 0, offsetRow = 0,
  margins = c(2.5, 4.5),
  lhei = c(1,7), lwid = c(2,5),

  # options of the graduated legend
  key.par=list(mar = c(3,2,2,2), mgp = c(1, 0.5, 0)),
  key.title = "", key.xlab = "Number of pollen grains",
  key.xtickfun=function() {return(list(at = c(0, 100, 250, 500, 750, 1000)/1000,
    labels = c(0, 100, 250, 500, 750, 1000)))}
)
```

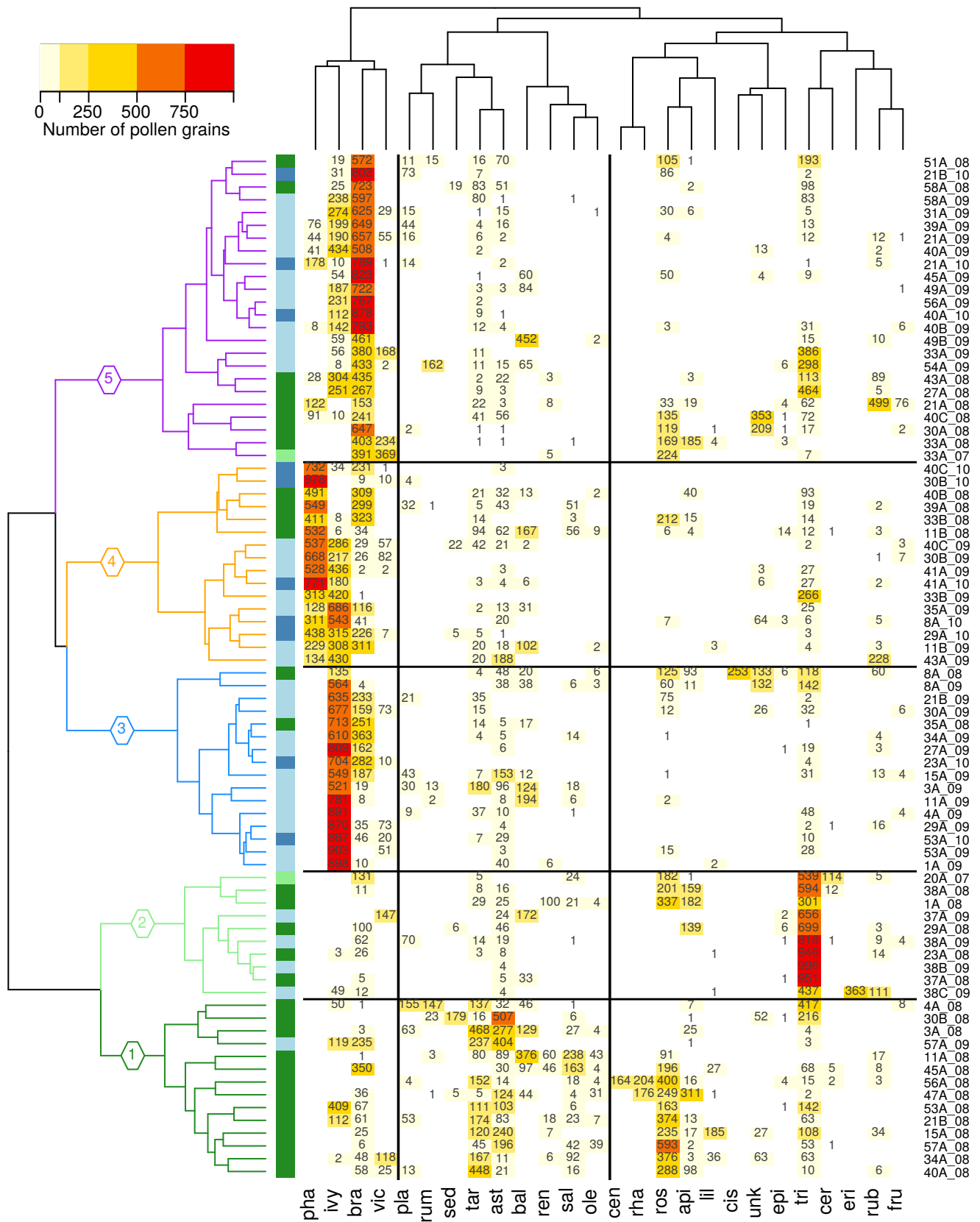


Figure 28:

Important note

With the last method, the clusters have been numeroted in the order they appear in the dendrogram which is easy and logical for discussion about these clusters. A problem that could appear is that if you use `cutree` to obtain the clusters from you dendrogram they will be numeroted in the order they appear in the dataset, not in the dendrogram. The groups will be the same but they arbitrary number will differ... You must use an alternative method to obtain the same group names/numbers (these groups will be used below in other cluster interpretation graphical methods).

```
# Create the groups with the classical approach --> these ones will not match the  
# groups in our heatmap...  
Groups5 <- factor(paste0("Grp", cutree(rowClust, k = 5)))  
  
# We use a slightly more complicated method just to have the same groups numbers as on  
# the last heatmap:  
Groups5 <- cutree(as.dendrogram(rowClust), k = 5, order_clusters_as_data = FALSE)  
Groups5 <- Groups5[order(order.dendrogram(as.dendrogram(rowClust)))]  
Groups5 <- factor(paste0("Grp", Groups5))
```

Alternative methods for heatmaps

The main drawbacks of `gplot::heatmap.2` are that :

- you can only add one supplementary variable
- there is no legend for the supplementary variable
- adding horizontal and vertical lines to separate the groups is not very straightforward

There are many other packages to do advanced heatmaps. Here are a few :

- `pheatmap` “Pretty heatmaps”. Easy option to separate the groups
- `d3heatmap` Interactive heatmap as html widgets. When you hover the mouse button on the heatmap, a popup window appears with the columns and row names and the value of the cell. You can also zoom in and out.
- `ComplexHeatmap` the most advanced package (from Bioconductor, not on the CRAN). Very powerful in particular when you want to add many supplementary variables (with their legends). The main drawback is that it is based on the

NB the installation process from bioconductor is different than the installation from CRAN. To install `ComplexHeatmap` you must proceed as follows :

```
# source("https://bioconductor.org/biocLite.R")  
# biocLite("ComplexHeatmap")
```

1.5.1.2 Visualize the clusters on a SPLOM

Each original plant species is a dimension and we can visualize the groups in these original dimensions with a ScatterPlot Matrix (SPLOM). This is however quite difficult for large datasets (many variables or many observations...).

Here we create a SPLOM of the 7 most abundant plant species with 5 clusters. Because the number of pollen grains is heavily left skewed (many small values, few high values) we applied a square root transformation to the data. A log transformation is another typical (but stronger) transformation in such situations. We could also represent the Hellinger transformed data instead. The most abundant plant species are the most important here because the clustering was performed on unscaled data. It would be much more difficult to find the interesting dimensions (species) to plot if the clustering was based on scaled data...

```
# dev.new(width = 18/2.54, height = 15/2.54)
pairs(sqrt(Y[,1:7]), gap = 0, oma=c(3,3,6,3), cex = 0.8,
      col = Groups5, pch = as.numeric(Groups5))
legend("top", legend = levels(Groups5), col = 1:5, pch = 1:5,
      xpd = NA, horiz = TRUE, bty = "n", inset = -0, pt.cex = 1)
```

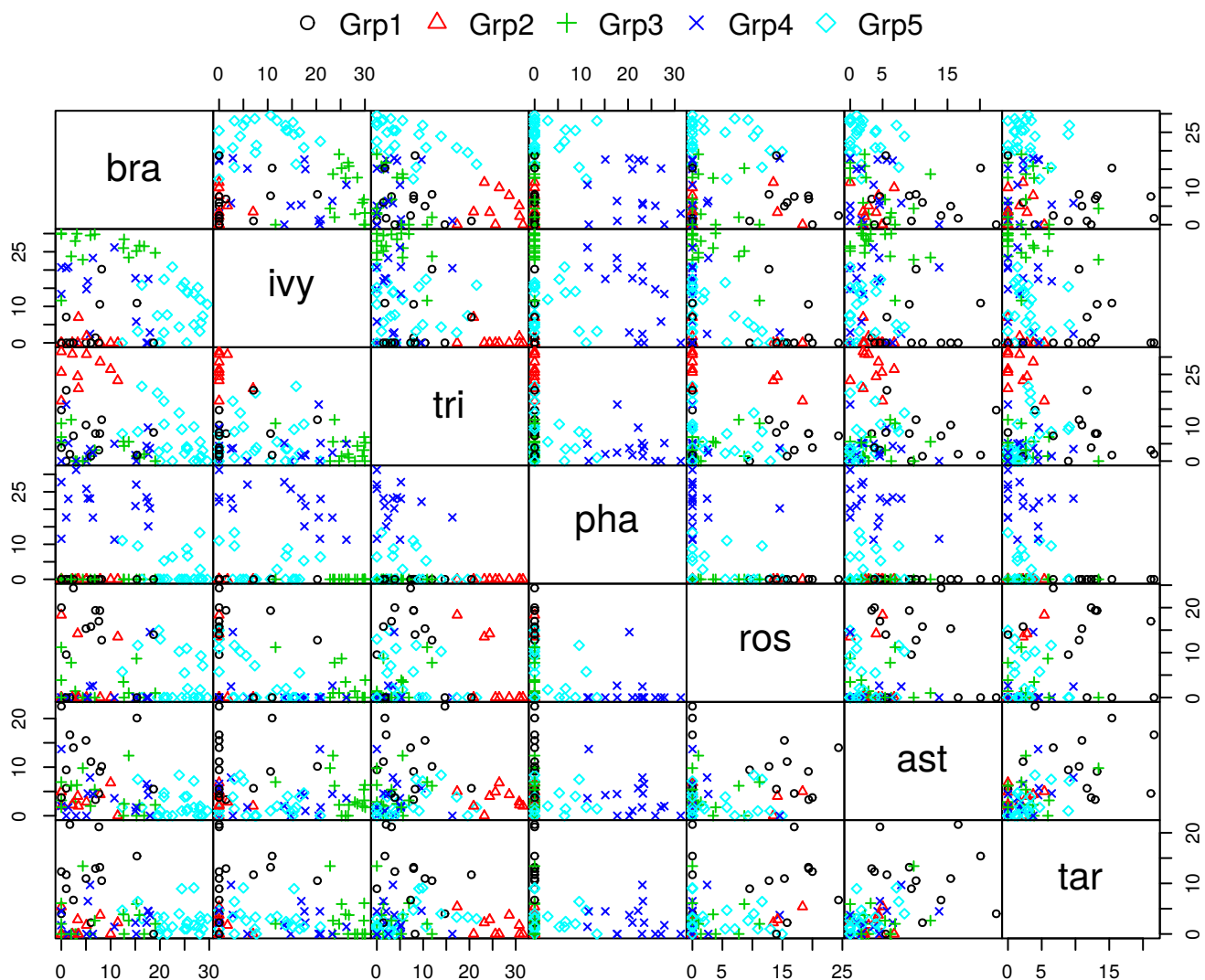


Figure 29:

1.5.1.3 Visualize the clusters on an ordination plot

One of the problems of the SPLOM is that it is difficult to use when you have many variables (here pollen species). Ordinations are typically used to decrease the number of dimensions needed to represent a dataset. The different clusters can then be visualized on an ordination plot. This is probably the most frequent approach for non hierarchical clustering. It might also be a good complement to heatmaps.

NMDS is usually the method that provides the best 2 dimensional solution (at least for visualisation purposes). However here 2 dimensions would not be enough to visualize this complex dataset. We represent then the groups with a 3 dimensions solution.

When the number of observations is too high and the plot over-crowded, an easy solution is to plot only a random sub-sample of the observations.

```
# dev.new(width = 18/2.54, height = 9/2.54)

NMDS <- MASS::isoMDS(hell_dist, k = 3)

## initial  value 24.199078
## iter    5 value 14.439741
## iter   10 value 13.526650
## final   value 13.488399
## converged

n <- 15 # how many species to display ?

par(mfrow = c(1,2), mar = c(3.5,3.5,1,1), mgp = c(2, 0.6, 0), cex = 0.8, las = 1)
choices = c(1,2)
plot(NMDS$points[,choices], asp = 1,
     pch = as.character(as.numeric(Groups5)), col = as.numeric(Groups5),
     xlab = paste0("NMDS dim ", choices[1]),
     ylab = paste0("NMDS dim ", choices[2]))
abline(v = 0, h = 0, col = "gray80", lty = 2)
spe <- wascores(NMDS$points[,choices], hell_trans)
ordilabel(spe[1:n,], labels = row.names(spe)[1:n], priority = n:1)

# same graph for dimensions 1 and 3
choices = c(1,3)
plot(NMDS$points[,choices], asp = 1,
     pch = as.character(as.numeric(Groups5)), col = as.numeric(Groups5),
     xlab = paste0("NMDS dim ", choices[1]),
     ylab = paste0("NMDS dim ", choices[2]))
abline(v = 0, h = 0, col = "gray80", lty = 2)
spe <- wascores(NMDS$points[,choices], hell_trans)
ordilabel(spe[1:n,], labels = row.names(spe)[1:n], priority = n:1)
```

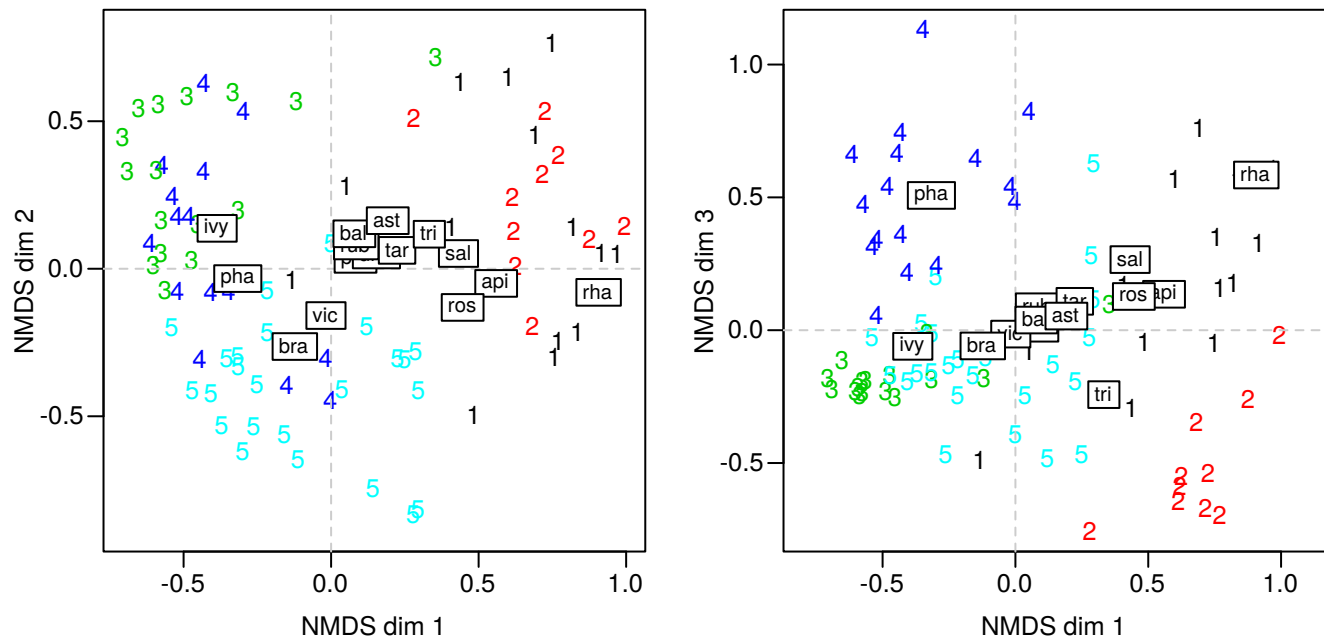


Figure 30:

```
# par(mfrow = c(2,3))
# for(k in 1:6){
#   NMDS <- isoMDS(hell_dist, k = k)
#   corr <- cor(dist(NMDS$points), hell_dist, method = "spearman")
#   plot(x = dist(NMDS$points), y = hell_dist)
#   title(paste0("Corr = ", round(corr,2)))
# }
```

1.5.2 Clusters description

Compute descriptive statistics (quantiles, average) etc... for each group and use graphs to visualise them

1.5.2.1 Describe the clusters with simple graphs

To understand what are the characteristics of each group, you might also simply compare the values of each variable between the clusters. You could compute simple descriptive statistics (eg the average of each pollen species for each group) but a graphical representation is almost always better.

Note : You might be tempted to perform some statistical tests here like an ANOVA (eg `aov(bra ~ Group, data = tmp)`). However the p-values you would obtain with such an approach would be mainly meaningless because the information contained in the pollen (eg `bra`) has already been used to create the clusters.

You might however always do that with external variables that have not been used in the clustering to create the clusters. For example you might perform a logistic regression to test if the sampling period is different between the groups (eg `glm(Period ~ Group, data = tmp, family = "binomial")`). The period was not used to create the clusters and the p-values would be valid. However you should use multiple testing correction if you do many comparisons (eg Bonferoni corection for a easy but very conservative method)

```
tmp <- Y
tmp$Group <- Groups5
tmp$Period <- d$Period
tmp$ID <- row.names(tmp)
tmp <- reshape2::melt(tmp, id = c("ID", "Period", "Group"),
                      variable.name = "Plant", value.name = "Nb")
```

Comparison of the 6 most abundant plants between groups

```
# dev.new(width = 18/2.54, height = 12/2.54)

ggplot(tmp[tmp$Plant %in% c("bra", "ivy", "tri", "pha", "ros", "ast"),],
       aes(y = Nb, x = Group)) +
  geom_boxplot() +
  geom_point(color = "gray75", alpha = 0.5, position = position_jitter(width = 0.1, height = 0))
  facet_wrap(~Plant) +
  theme_bw()
```

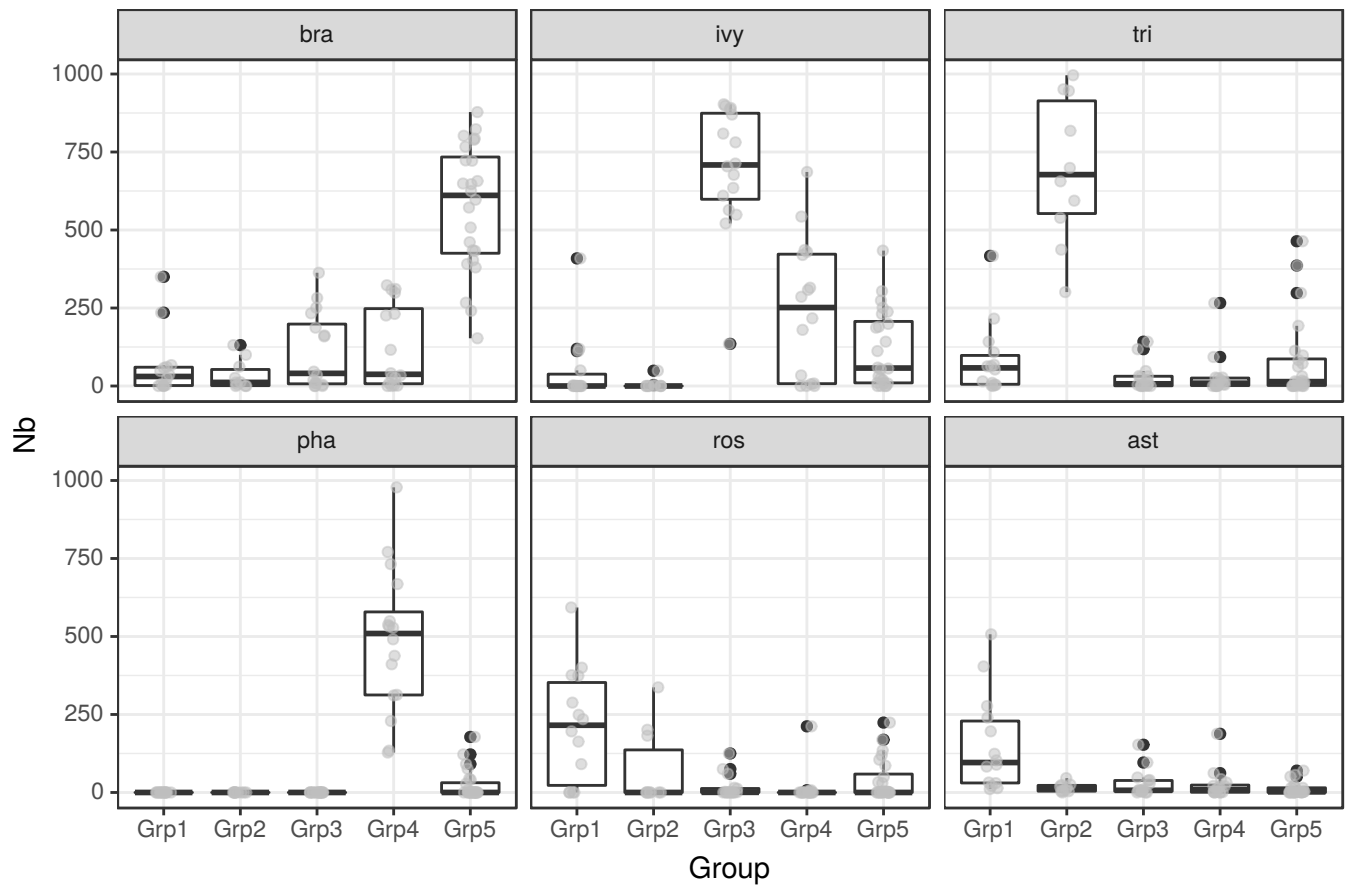



Figure 31:

Another way to look at the same data ...

```
# dev.new(width = 18/2.54, height = 12/2.54)
ggplot(tmp[tmp$Plant %in% colnames(Y)[1:15],],
  aes(y = Nb, x = Plant)) +
  geom_boxplot()+
  # geom_point(alpha = 0.5, position = position_jitter(width = 0.1, height = 0)) +
  facet_wrap(~Group, ncol = 2) +
  # scale_y_continuous(trans = "sqrt") + # uncomment this to have a square root scale
  theme_bw()
```

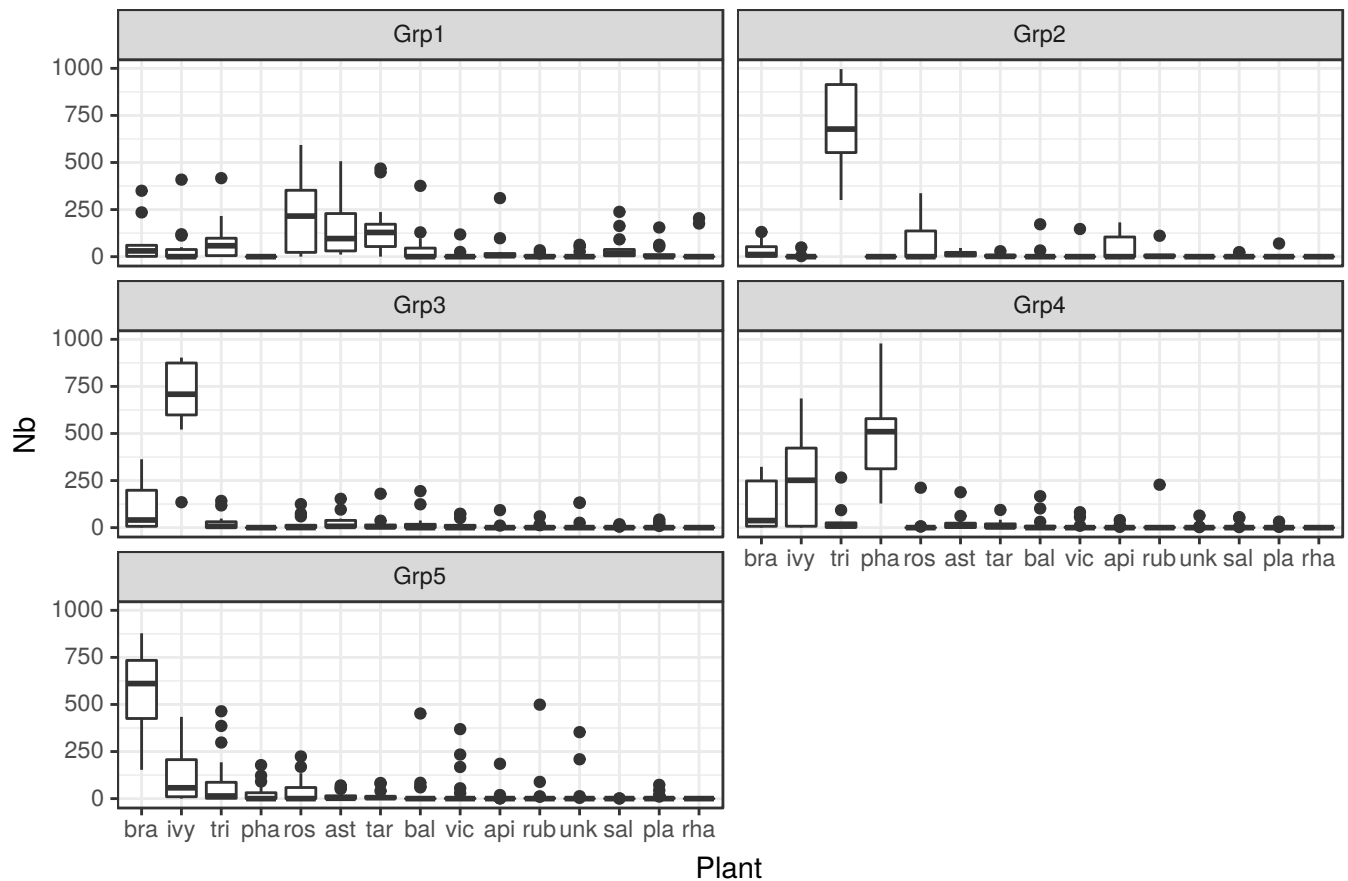


Figure 32:

Testing if the period differs between groups with a binomial GLM (= logistic regression). Then we perform all pairwise post-hoc comparisons (with p-value correction) and represent the predicted probabilities on a graph with a compact letters display (different letters means that the groups differences are statistically significant).

Here all the groups are significantly different from each other with Group 1 clearly dominated by July/August samples, group 2 and 3 dominated by September/October samples and Groups 4 and 5 more intermediate.

```
m <- glm(Period ~ Group, data = tmp, family = "binomial")
library(multcomp)
library(visreg)
mc <- glht(m, linfct = mcp(Group = "Tukey"))
summary(mc)
```

```
##
## Simultaneous Tests for General Linear Hypotheses
##
## Multiple Comparisons of Means: Tukey Contrasts
##
##
## Fit: glm(formula = Period ~ Group, family = "binomial", data = tmp)
##
## Linear Hypotheses:
##           Estimate Std. Error z value Pr(>|z|)
## Grp2 - Grp1 == 0    2.1595     0.2397   9.010 < 1e-04 ***
```

```
## Grp3 - Grp1 == 0  4.5109      0.2518  17.915 < 1e-04 ***
## Grp4 - Grp1 == 0  3.6636      0.2329  15.730 < 1e-04 ***
## Grp5 - Grp1 == 0  3.0758      0.2197  14.001 < 1e-04 ***
## Grp3 - Grp2 == 0  2.3514      0.1949  12.062 < 1e-04 ***
## Grp4 - Grp2 == 0  1.5041      0.1698   8.856 < 1e-04 ***
## Grp5 - Grp2 == 0  0.9163      0.1512   6.060 < 1e-04 ***
## Grp4 - Grp3 == 0 -0.8473      0.1865  -4.542 < 1e-04 ***
## Grp5 - Grp3 == 0 -1.4351      0.1698  -8.454 < 1e-04 ***
## Grp5 - Grp4 == 0 -0.5878      0.1402  -4.192 0.000258 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## (Adjusted p values reported -- single-step method)

# plot(mc)
# cld(mc)

# dev.new(width = 8/2.54, height = 8/2.54)
par(mar = c(3.5,3.5,3,1), mgp = c(2, 0.6, 0), cex = 0.8, las = 1)
visreg(m, scale = "response", rug = FALSE,
       ylab = "Probability to be a September/October sample")
mtext(text = cld(mc)$mcletters$Letters, line = 1, side = 3, at = ((0:4)+0.5)/5, cex = 0.8)
```

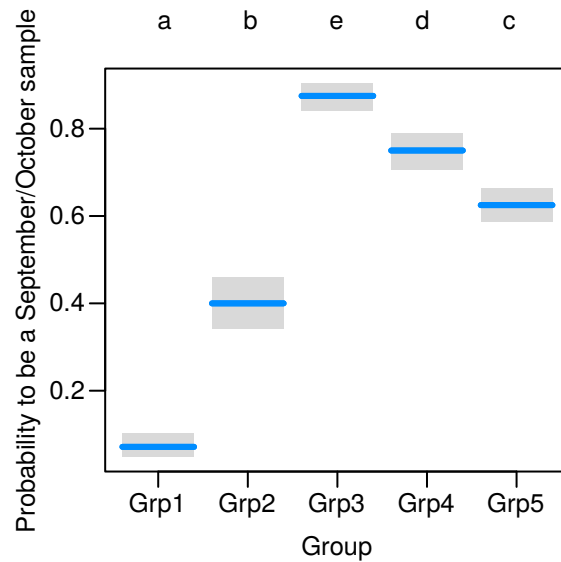


Figure 33:

1.5.2.2 Describe the clusters with pseudo-supervised approaches (eg : classification tree)

Again this approach is rather circular : we will try to find the variables that predict/explain the clusters as well as possible but the same variables have been used to create the clusters in the first place (this is why I call it “pseudo-supervised”).

The aim here is purely descriptive (the aim is not to validate the clusters created).

Classification trees are easy to use and interpret and seem well suited for such a task. But you might use any other regression/classification like method for which the interpretation is relatively easy and straight forward (eg : logistic regression = binomial or multinomial GLM, Linear Discriminant Analysis, ...). Note that if these methods provide p-values they are not valid here. Cross-validation is also useless here (because the test sets are never independent from the training set). So there is no way to “prune the tree” as is usually done with classification trees.

Note that the classification tree method used here must not be confounded with a hierarchical clustering approach. The classification tree is normally a supervised method - you partition one matrix based on thresholds from another matrix of “explanatory variables”. The hierarchical clustering is an unsupervised approach.

The classification trees have one major drawback : completely different trees might separate the clusters (almost) as well as the tree provided by the `rpart` function. This will be particularly frequent when the variables are highly correlated. So these classification trees generally give you only a partial view of the characteristics of the clusters.

```
library(rpart)
library(rpart.plot)

tmp <- Y
tmp$Group <- factor(paste0("Grp", cutree(rowClust, k = 5)))
m <- rpart(Group ~., data = tmp)
rpart.rules(m)

## Group  Grp1 Grp2 Grp3 Grp4 Grp5
## Grp1 [ .78 .06 .00 .06 .11] when bra < 372 & pha < 125 & ivy < 465 & tri < 427
## Grp2 [ .00 1.00 .00 .00 .00] when bra < 372 & pha < 125 & ivy >= 465
## Grp3 [ .00 .00 1.00 .00 .00] when bra < 372 & pha >= 125
## Grp4 [ .00 .00 .00 .90 .10] when bra < 372 & pha < 125 & ivy < 465 & tri >= 427
## Grp5 [ .00 .00 .00 .00 1.00] when bra >= 372

# simple graph
# prp(m, extra = 1, branch = 1) # basic graph

# improved graph

# dev.new(width = 12/2.54, height = 10/2.54)

prp(m, extra=1, branch=1, cex=0.7, digits=1,
     round=1, split.cex=1.1, split.round=.5, under = FALSE,
     split.box.col="lightgray", split.border.col = "darkgray",
     yesno.yshift = 0.6, xsep=" | ")
```

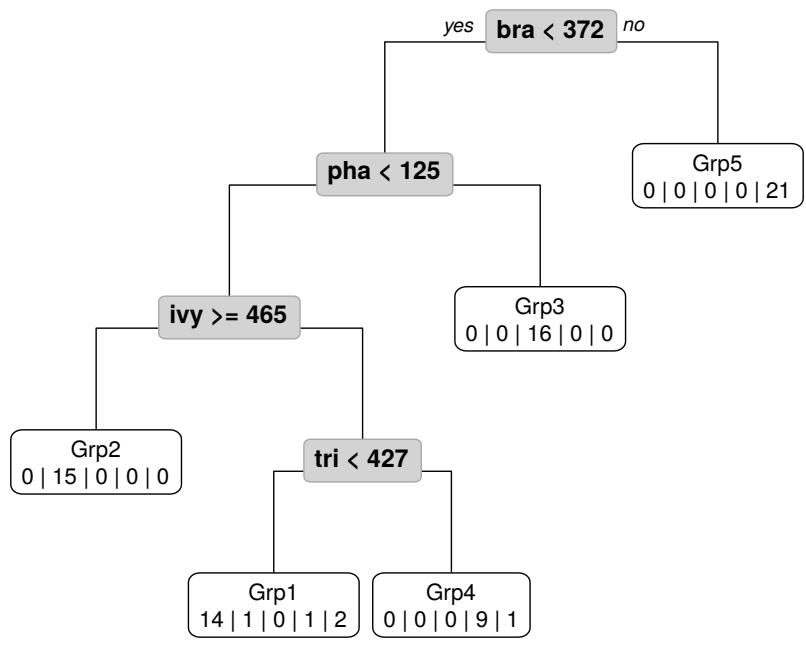


Figure 34:

1.6 Clustering validation

The aim of clustering validation is to assess the “quality” of the clusters, ie to answer the following questions :

1. How “good” are my clusters ?
2. How many clusters should I use ?
3. Which clustering method works “best” ?
4. Are the clusters “real” or am I creating clusters from random noise ?

The problem, as we have seen is that there is no easy answers to these question because the definition of a “good” cluster will be different for different persons, different datasets, different questions.

Each discipline has developped **specific ways to validate/characterise** the clusters. For example, genomics, phylogenetics and community ecology tend to use methods that are particularly adapted to the biological aspects related to their clustering : indicator species, genes function, phylogeny stability,...

There are however some descriptive statistics (clustering quality indices) and approaches that can be more broadly applied.

Typically we can use **internal criteria** ie criteria based on the information present only within the cluster and dataset. Ideally, the clusters should be :

1. Interpretable : via visual inspection and expert knowledge eg heatmaps (less easy for huge datasets)
2. Compact : low distance between observations from the same cluster, no within cluster gaps
3. Well separated : high distance between clusters, no intermediate points
4. Stable : small perturbations in the data should produce similar clusters
5. Robust : different clustering methods provide similar clusters on the same data. Clustering of the same objects with different variables (eg a plant dataset and a insect dataset) provides similar clusters,...

Many indices have been developed to characterize some of these criterion but depending on the criterion used the quality of the clusters and the number of clusters chosen might be very different...

You can also use **external criteria** : are these clusters congruent with external information that was not used to create the clusters ? eg : do the clusters match the sampling structure, temporal or spatial structure, groups that have not been used in the clustering,... Has the clustering good predictive for an external data of interest ?

Frequently the answer to these question will be “it depends”. For example in community ecology the question of “how many clusters should I use” is often difficult to answer because of the frequent highly hierarchical structure of communities. You might for example have 2 main clusters that will differentiate vegetation communities from forests and grasslands. These two groups will usually be very well separated. But of course you might divide further the “forest” cluster into deciduous forests, coniferous forests and mixed forests that might be less well separated (intemediate conditions) but that can be nevertheless relevant. And for example the deciduous forests can be even further separated in smaller groups depending on the underground, humidity, climate,... Where you decide to stop is your decision based on the interpretability of the clusters even if a clustering quality index tells you that 2 clusters is the best solution...

Sometimes clusters that are not well separated and not very compact might still be a usefull description and simplification of reality.

In the end the interpretability of the cluster must be your main guide and the clustering validation indices detailed below should just help you in your decision but they should not be the only criterion used...

Clustreing validation is a complex and growing research topic. A good starting point if you want to dig deeper are several R packages dedicated to cluster validation with their associated vignettes or papers : `NbClust`, `clValid`, `optCluster`, `pvclust`... The package `cluster` contains also several usefull functions

for cluster validation. you need to take the time to understand how these methods work because they are not always applicable to all datasets (eg `pvclust` and the stability indices of `clValid`).

1.6.1 Silhouette width

The silhouette width is a very widely used clustering validation index. It has several advantages : it is easy to understand, easy to interpret, usable for any clustering algorithm, and it can provide an estimation of the clustering quality for each observation, each cluster (average of the individual silhouettes of one cluster) or for the whole clustering solution.

Here we will just present the index and its use when you have chosen one algorithm and a number of clusters. However, it can also be computed to compare different number of clusters and different algorithms (see in the next sections).

It is a combined measure of the separatedness and the compactness of the clusters.

The computation works as follows :

1. compute A, the average distance between one point and all the points from the same cluster
2. compute B, the average distance between this point and all the points of his nearest cluster
3. the silhouette width for this point is : $S = (A - B) / \max(A, B)$

You can then also estimate the average silhouette width for each cluster and for all clusters.

Interpretation :

- the values are comprised between -1 and 1 and should be maximized
- a silhouette width < 0 means that the point is on average more close to the point of the nearest cluster than to the points of his own cluster \rightarrow it is probably missclassified. . .
- a silhouette width = 0 means that the point is intermediate between two clusters.
- a silhouette width > 0 means that the point is well separated from his cluster

A drawback of this approach is that clusters with few points tend to have higher silhouette widths.

The basic code to compute silhouette width and produce a graphical representation is quite straightforward (NB the code is not run here) :

```
dist_matrix <- dist(scale(iris[,1:4])) # Euclidean distance matrix
hcl <- hclust(dist_matrix, method = "ward.D2") # compute a hierarchical clustering
groups <- cutree(hcl, k = 10) # cut the tree to extract the groups
sil_width <- cluster::silhouette(groups, dist = dist_matrix) # compute the silhouette width
plot(sil_width, border=NA) # plot
```

With a little more work you can however greatly improve the visualisation. We use a fake dataset of random uniform numbers for 2 variables x and y. Here we add numbers and colors to the groups. We also highlight the points with low silhouette width (< 0.1) with gray squares on the x y full representation of the data (NB in real life dataset you could do that on the results of an ordination). And we spot the same points on the dendrograms by adding red rectangle along the x axis.

```

# fake random dataset
set.seed(12)
tmp <- matrix(runif(1000), ncol = 2)
colnames(tmp) <- c("x", "y")

k = 4
mycols <- c("forestgreen", "dodgerblue", "orange", "purple")

# Compute the clustering and transform into a dendrogram object with numbered groups
hcl <- hclust(dist(tmp), method = "ward.D2")
hcl_colored <- color_branches(as.dendrogram(hcl), col = mycols[1:k],
                             groupLabels=TRUE, k = k)

# create the groups in the same order as the dendrogram
groups <- cutree(hcl_colored, k = k, order_clusters_as_data = FALSE)
groups <- groups[order(order.dendrogram(hcl_colored))] # to match the order of the data

# Compute the silhouette width
sil_width <- cluster::silhouette(groups, dist = dist(tmp))

# Spot the observations with a silhouette width < 0.1
# Then create a vector of red colors for these point with a "bad" silhouette width
bad_sil_width <- sil_width[, "sil_width"] < 0.1
sil_width_colors <- c(NA, "orangered")[as.numeric(bad_sil_width)+1]
sil_width_colors <- sil_width_colors[order.dendrogram(hcl_colored)] # reorder like the dendrogram

# x y Graph with the points with a silhouette width < 0.1 underlined by a gray square
# dev.new(width = 18/2.54, height = 9/2.54)
par(mfrow = c(1,2), mar = c(2.5,2.5,2,1), mgp = c(1.5, 0.5, 0), cex = 0.8, las = 1)
plot(tmp[, c("x", "y")], col = mycols[groups], pch = groups,
     main = paste0("Ward.D2 - ", k, " groups"))
points(tmp[bad_sil_width, c("x", "y")], pch = 0, col = "gray50", cex = 2)

# Plot the dendrogram with groups and add a red rectangle label for the observations with
# a silhouette width < 0.1
plot(hcl_colored, main = paste0("Ward.D2 - ", k, " groups"), leaflab = "none")
# rect.hclust(hcl, k = k, border = mycols[1:k])
symbols(x = 1:nrow(tmp), y = rep(-8*max(hcl$height)/100, nrow(tmp)),
        rectangles= cbind(rep(1, nrow(tmp)), max(hcl$height)/8),
        bg= sil_width_colors,
        fg = NA, add=TRUE, inches=FALSE)

```

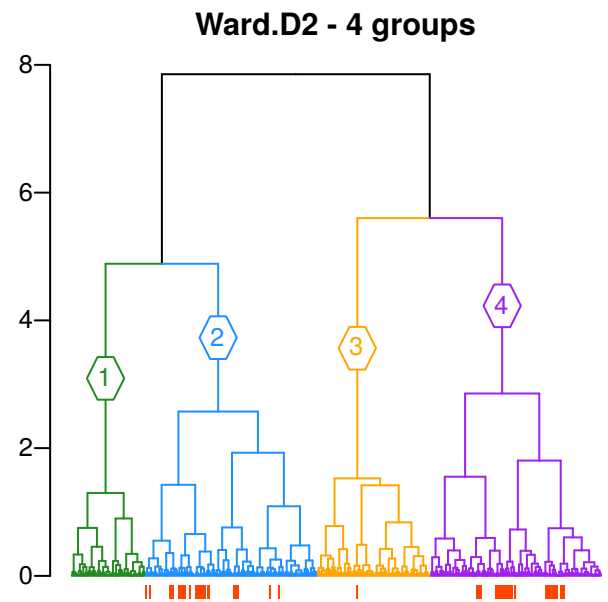
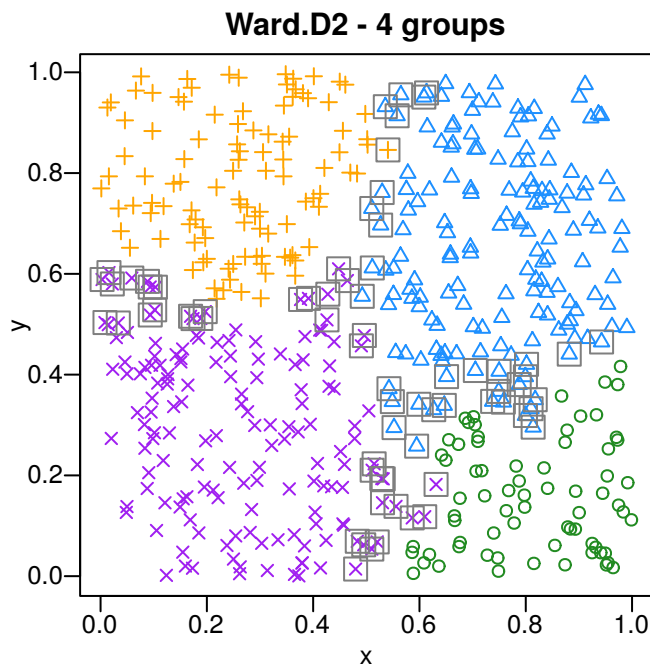



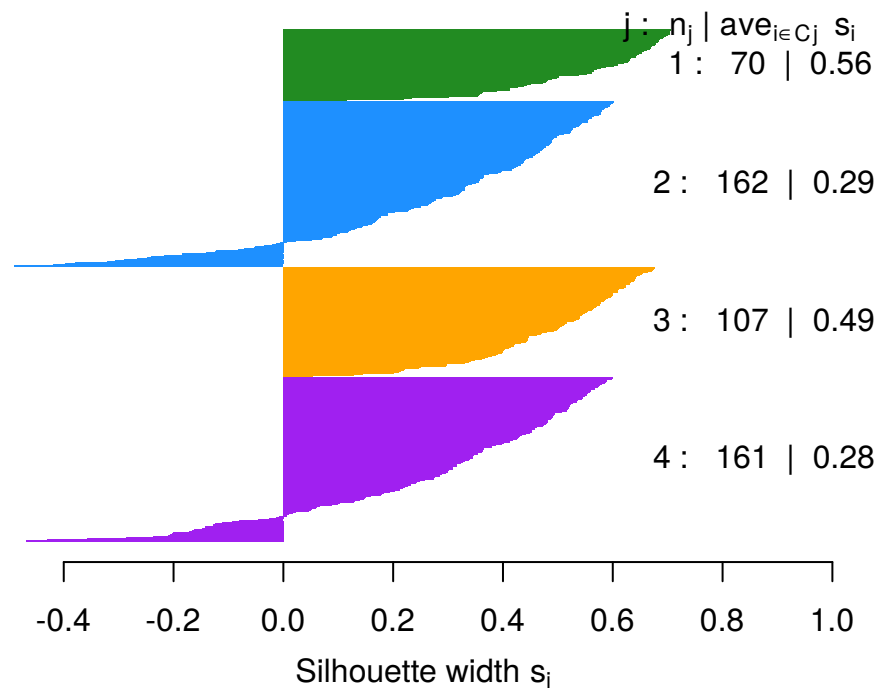
Figure 35:

```
# Plot the silhouette widths
```

```
# dev.new(width = 14/2.54, height = 12/2.54)
```

```
plot(sil_width, col = mycols[1:k], border = NA, do.n.k = FALSE, main = "Silhouette Plot")
```

Silhouette Plot



Average silhouette width : 0.37

Figure 36:

1.6.2 Choose the “best” number of clusters with the Gap statistic

Computes the difference between the within cluster variation of the real dataset and the same value computed under a “null hypothesis” dataset without clustering (for each variable : uniformly distributed random numbers between the min and the max of that variable). The process is repeated several times (default = 100) to obtain a standard error of the statistic. This is repeated for several k numbers of clusters

The higher the gap statistic the better the clustering. However the value tends to increase with higher number of clusters. The one standard deviation rule is generally applied : $\text{Gap}_k \geq \text{Gap}_{k+1} - \text{se}(\text{Gap}_{k+1})$. The function `cluster::maxSE` can find this value for you...

This approach has several disadvantages : the computation is slow, the interpretation of the gap statistic is not very intuitive (imho),...

A major advantage however is that many other clustering statistics need at least two clusters to be computed. So when the best solution is “2 clusters” you don’t know if this is because you really have 2 clusters or because there is no clustering structure in the dataset. The gap statistic can be computed for 1 cluster and if the best solution found is “1 cluster”, it means that there is probably no clustering structure in the dataset.

In the following example we see that for the iris dataset (ward.D2 hclust) the “best” number of clusters would be 2 while for the random dataset the best number would be 1 which means that there is no clustering structure in this dataset (which is expected because this is a random uniform dataset...)

```
# Gap statistic on the iris dataset
```

```
# dev.new(width = 16/2.54, height = 8/2.54)
par(mfrow = c(1,2), mar = c(2.5,3,3,1), mgp = c(1.8, 0.5, 0), cex = 0.8, las = 1)
mycut <- function(x, k) {list(cluster = cutree(hclust(dist(x), "ward.D2"), k = k))}
gap <- cluster::clusGap(scale(iris[,1:4]), mycut, K.max = 10)
best_gap <- cluster::maxSE(gap$Tab[, "gap"], gap$Tab[, "SE.sim"])
plot(gap, main = paste0("Gap Statistic iris data \nOptimal cluster number = ", best_gap))
abline(v = best_gap, lty = 2, col = "gray50")

# gap statistic on a random dataset
gap <- cluster::clusGap(tmp, mycut, K.max = 10)
best_gap <- cluster::maxSE(gap$Tab[, "gap"], gap$Tab[, "SE.sim"])
plot(gap, main = paste0("Gap Statistic random data \nOptimal cluster number = ", best_gap))
abline(v = best_gap, lty = 2, col = "gray50")
```

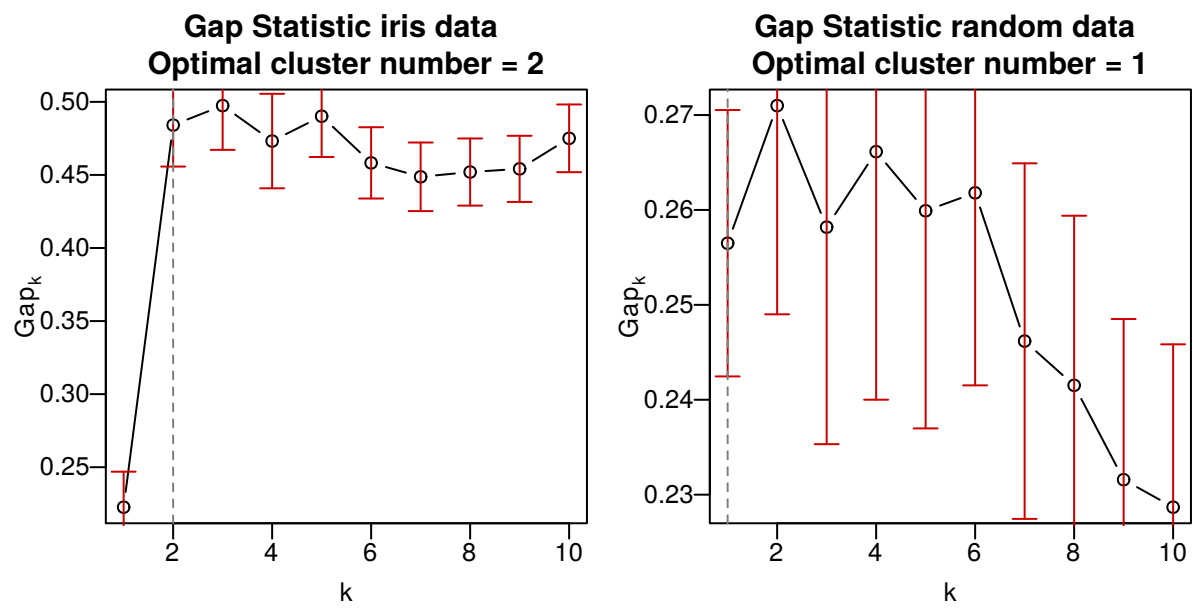


Figure 37:

1.6.3 Choose the “best” number of clusters with NbClust

A widely recognized problem with the many clustering validation statistics available is that almost each statistic will give you a different answer about the “best number of clusters” to use. The idea of the NbClust package is to compute ~ 30 of these indices (including the gap statistic and the silhouette width) and let them “vote” for the best number of clusters.

This is a very coarse approach. It can be applied only to one clustering algorithm at a time and only to hierarchical clustering (with the different grouping methods of hclust) and k-means clustering. It is difficult to use it to compare different algorithms (however you can get the results of all the computations). You also have the “best” number for each statistic but sometimes the next best number might be almost as good.

The function also automatically prints a lot of messages that are impossible to suppress which is painful...

It can be however a very easy first step to have a rough estimate of a reasonable number of clusters in an interactive session ...

Example with the iris dataset and Hierarchical clustering with ward.D2 method. Most of the indices “vote” for the 2 clusters solution but the 3 clusters solution is also quite popular.

```
iris_scaled <- scale(iris[,1:4])
iris_dist <- dist(iris_scaled)

library(NbClust)
nb <- NbClust(data = iris_scaled, diss = iris_dist, distance = NULL,
              method = "ward.D2", min.nc = 2, max.nc = 15, index = "alllong")

## *** : The Hubert index is a graphical method of determining the number of clusters.
##           In the plot of Hubert index, we seek a significant knee that corresponds to a
##           significant increase of the value of the measure i.e the significant peak in Hu
##           index second differences plot.
##
## *** : The D index is a graphical method of determining the number of clusters.
##           In the plot of D index, we seek a significant knee (the significant peak in Din
##           second differences plot) that corresponds to a significant increase of the valu
##           the measure.
##
## *****
## * Among all indices:
## * 12 proposed 2 as the best number of clusters
## * 6 proposed 3 as the best number of clusters
## * 1 proposed 4 as the best number of clusters
## * 3 proposed 5 as the best number of clusters
## * 2 proposed 7 as the best number of clusters
## * 4 proposed 15 as the best number of clusters
##
##           ***** Conclusion *****
##
## * According to the majority rule, the best number of clusters is 2
##
##
## *****
```

You can also compute the results for the k-means algorithm. Here both the 2 and 3 clusters solutions are equally popular.

```
nb <- NbClust(data = iris_scaled, diss = iris_dist, distance = NULL,
              method = "kmeans", min.nc = 2, max.nc = 15, index = "alllong")

## *** : The Hubert index is a graphical method of determining the number of clusters.
##           In the plot of Hubert index, we seek a significant knee that corresponds to a
##           significant increase of the value of the measure i.e the significant peak in Hu
##           bert index second differences plot.
##
##
## *** : The D index is a graphical method of determining the number of clusters.
##           In the plot of D index, we seek a significant knee (the significant peak in Din
##           second differences plot) that corresponds to a significant increase of the valu
##           the measure.
##
## *****
## * Among all indices:
## * 10 proposed 2 as the best number of clusters
## * 10 proposed 3 as the best number of clusters
## * 1 proposed 5 as the best number of clusters
## * 2 proposed 10 as the best number of clusters
## * 5 proposed 15 as the best number of clusters
##
##           ***** Conclusion *****
##
## * According to the majority rule, the best number of clusters is 2
##
## *****
```

1.6.4 Compare different algorithms and different number of clusters with the silhouette width and Dunn index

The `mytoolbox.R` script contains a wrapper function `NbClust2` based on `NbClust` but that facilitates the comparison of different algorithms and different number of clusters.

It uses only the silhouette width and the Dunn index. The Dunn index is also a widely used index that is quite easy to understand and interpret. It has a value between 0 and infinity and it should be maximized. It is the ratio between the between cluster distance (separation) and within cluster maximum distance (compactness).

It can be computed as follows :

- compute A, the minimum distance between the points of the cluster and the points of the other clusters
- compute B, the maximum distance between the points within that cluster
- The Dunn index is the ratio A/B

Its drawbacks is that it can be computed only for each cluster or for all clusters but not for each point. Another drawback is that the Dunn index is based on only two pairs of observations for each cluster so it is quite sensitive to outliers.

The `NbClust2` function will compute the silhouette width and Dunn index for different algorithms (k-means or `hclust` with any grouping method) and different number of clusters. The results can easily be plotted with a dedicated plotting method based on `ggplot`.

You can also perform the computation on different distance matrices at once for `hclust` and for different (transformed) datasets for k-means.

Simple example with default values for the iris dataset. The average linkage and `ward.D` algorithms for two clusters seem promising...

```
d <- scale(iris[, -5])
res <- NbClust2(dist(d))

# dev.new(width = 16/2.54, height = 8/2.54)
plot(res)
```

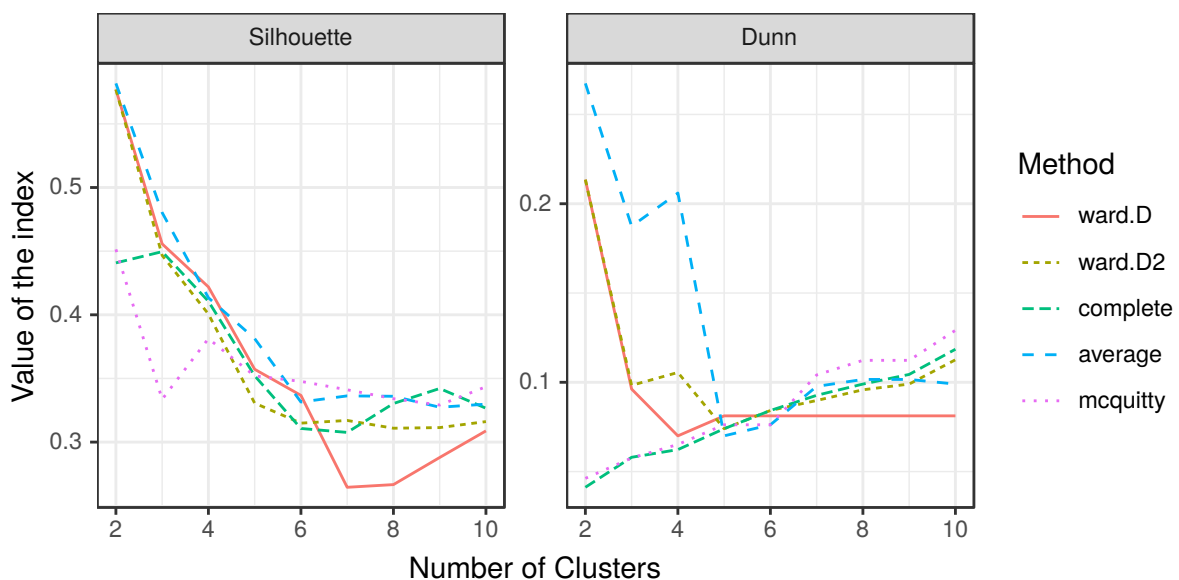


Figure 38:

```
# you can choose to plot only one of the indices
# plot(res, index = "Dunn")
# plot(res, index = "silhouette") # index name is case insensitive
```

More complex example.

We use here 3 distance matrices for the hierarchical clustering methods. We also add kmeans with `dokmeans = TRUE` on the scaled and unscaled dataset. To compute kmeans you need to provide the transformed or not transformed datasets (the distance matrix is ignored for k-means, they are only used by `hclust`).

Note that the indices are comparable between different methods within a given distance matrix (`hclust`) or a given dataset (`kmeans`). You cannot compare for example the silhouettes values between the scaled and unscaled dataset because the distances are not comparable (due to the different scales).

```
res <- NbClust2(diss = list(scaled = dist(scale(iris[,-5])),
                           unscaled = dist(iris[,-5]),
                           manhattan = dist(scale(iris[,-5]), method = "manhattan")),
               kmax = 10, dokmeans = TRUE,
               data = list(scaled = scale(iris[,-5]), unscaled = iris[,-5]))
```

```
# dev.new(width = 18/2.54, height = 12/2.54)
plot(res)
```

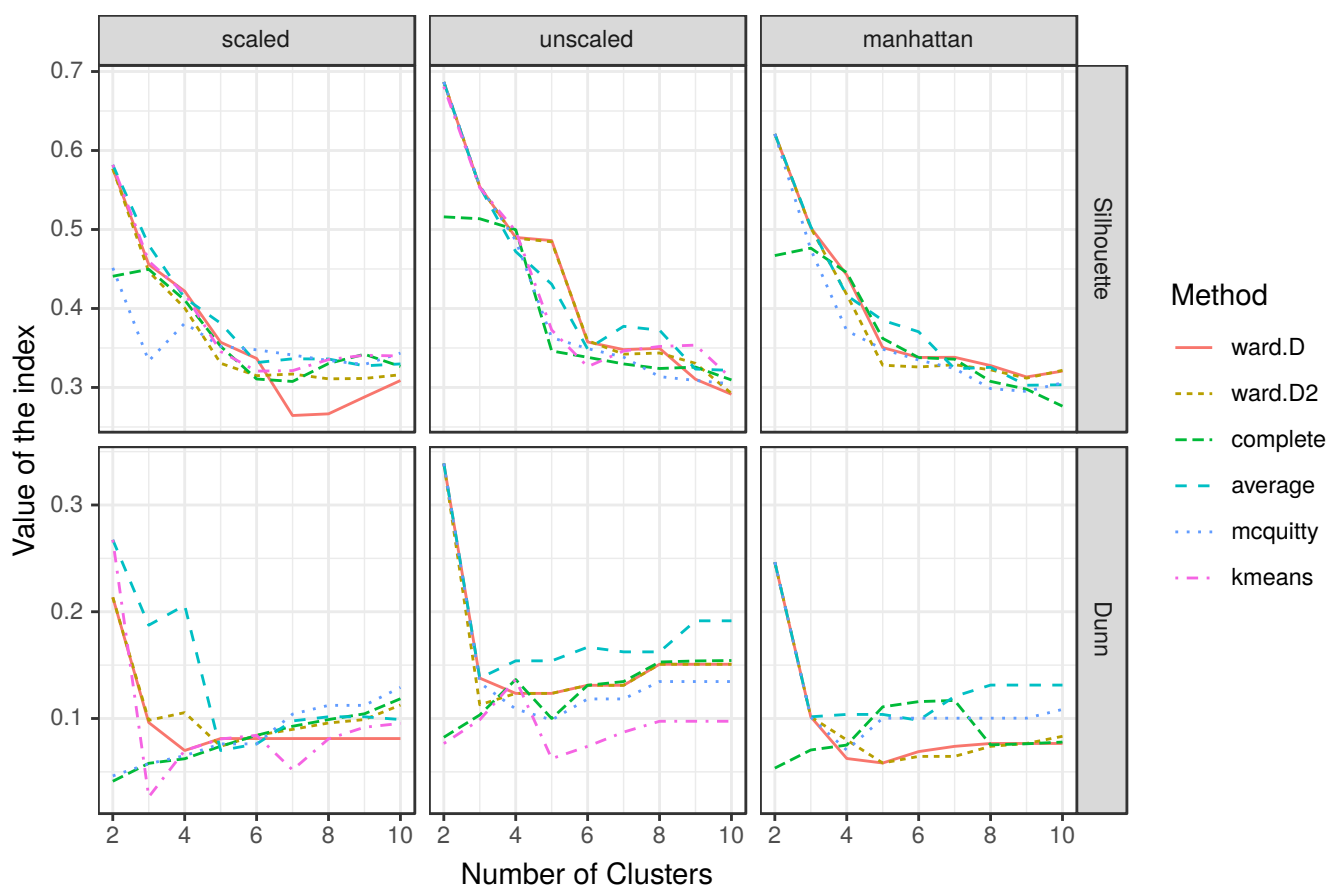


Figure 39:

A final example : we use here the pollen dataset for which we produced a heatmap before. We compute the indices on 4 distance matrices : Hellinger, Euclidean, Bray-Curtis and Jaccard (after binary transformation). We also compute the kmeans solution for the Hellinger transformed and Euclidean datasets.

Remember that you cannot compare the indices between the distance matrices. The silhouette width of the Euclidean distance based clustering are higher than the ones based on the hellinger transformed dataset but it does not mean that the euclidean approach is better...

For the hellinger distance, the k-means and ward.D2 solutions for 5 clusters look promising. For the Bray-Curtis distance, 5 to 7 clusters with the 2 Wards algorithms seem promising.

Note that the average linkage tends to give higher values for the highest number of clusters. This is probably due to the creation of very small clusters with few observations that have the tendency to increase the silhouette width (you might check this on a dendrogram).

All average silhouette widths are > 0 . On average the points are closer to the points of their own cluster than to the points of the other clusters...

Note that the Dunn results are completely different and quite difficult to interpret here... The fact that the Dunn index is always < 1 means that the minimum distance between clusters is on average smaller than the maximum distance between clusters. We have large clusters and their borders are close to each other...

The next step after a graph like that would probably be to examine the dendrograms and heatmaps of the most promising solutions and check which one gives you the most useful and interpretable solution ...

```
# reread the dataset
d <- read.csv2("data/pollen/full_dataset.csv")
d <- data.frame(d[, c(2:5)], d[, 14:47])
d <- unique(d)
d <- na.omit(d)
d <- d[,c(1:4, which(colSums(d[, -c(1:4)]) > 50) + 4)]
# select the columns to use
Y <- d[, -c(1:4)]

hell_trans <- decostand(Y, "hellinger") # hellinger transformation
hell_dist <- dist(hell_trans) # hellinger distance between the rows (samples)

res <- NbClust2(data = list(euclid = Y, hellinger = hell_trans), # used only to compute k-means
               diss = list(hellinger = hell_dist, euclid = dist(Y),
                           braycurtis = vegdist(Y, method = "bray"),
                           jaccard = vegdist(decostand(Y, "pa"), "jaccard")),
               kmax = 14,
               method = c("ward.D", "ward.D2", "complete", "average", "kmeans"))

# dev.new(width = 18/2.54, height = 12/2.54)
plot(res)
```

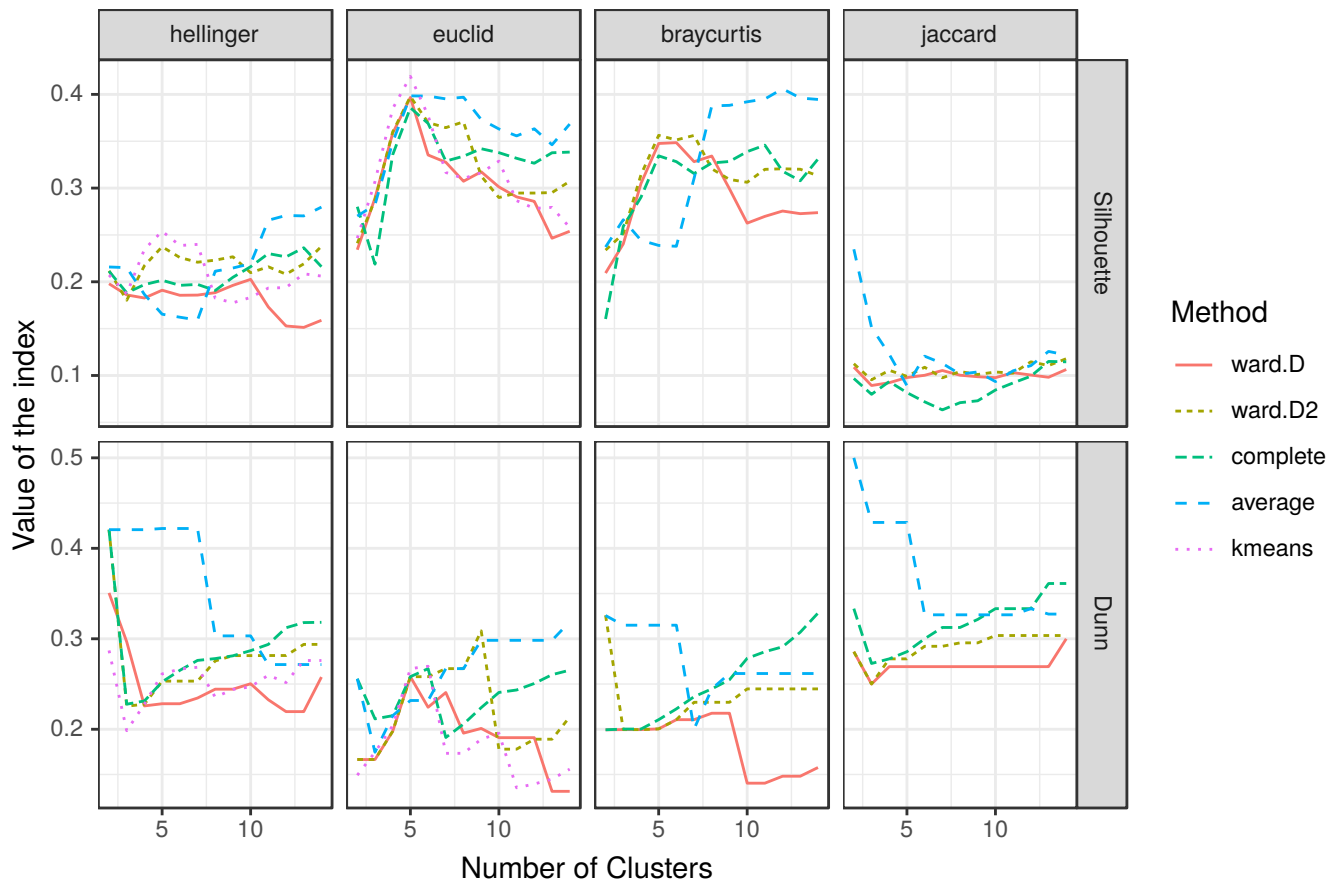



Figure 40:

1.6.5 Compare different algorithms and different number of clusters with `clValid`

The `ClValid` package offers very interesting cluster validation approaches mainly for genomic data. It can compare many different clustering algorithms (but only one grouping method at a time for `hclust`). However the only distance possible are Euclidean, correlation and Manhattan. (so impossible to use for Jaccard, Bray-Curtis,...)

It provides 3 types of validation indices :

1. `internal` : including the silhouette width, dunn index and a connectivity index. The connectivity index is comprised between 0 and infinity and should be minimized. It checks if the nearest neighbours of a point are in the same clusters as this point. If the second nearest neighbour of a point is not in the same cluster it will increase the index by a value of $1/2$, if the 5th nearest neighbour is not in the same cluster, it will increase the index by $1/5$, etc.
2. `stability measures` : a series of measures performed after removing one of the variables. Note that this might be interesting for dataset with highly redundant variables (like omics datasets) but it will probably be meaningless for species community data : if you remove the dominant species of a community of course you will obtain a very different clustering solution. Removing a variable from the dataset might be a small perturbation for omics datasets (because of the redundancy) but it will often not be a small perturbation for sites x species datasets.
3. `biological measures` : mainly based on gene function databases and so valid only for genomic data when this information exists

Example on the hellinger transformed pollen data with `ward.D` `hclust`, `k-means` and `pam`. You need to provide the raw or transformed data and a distance matrix will be computed internally (Euclidean by default).

Note that based on the results of the previous section, the `ward.D2` algorithm seems to provide much better results than the `ward.D` algorithm which is the one used here for `hclust` (`clMethods = "hierarchical"`) and `clValid` won't allow you to use the `ward.D2` grouping method with `hclust`.

However if you want to visualize an equivalent to the `ward.D2` results you can add `clMethods = "agnes"`. The default `ward` grouping method of `agnes` (ie the function from package `cluster` that performs hierarchical clustering just like `hclust`) is equivalent to the `ward.D2` method of `hclust`. If you use other grouping methods (like "average") `clMethods = c("hierarchical", "agnes")` will give you the same results (avoid that to avoid overcrowded graphs...)

```
library(clValid)
intern <- clValid(hell_trans, 2:15,
                 clMethods = c("hierarchical", "kmeans", "pam"),
                 method = "ward",
                 validation = "internal")

# dev.new(width = 22/2.54, height = 7/2.54)
# summary(intern)
par(mfrow = c(1,3), mar = c(3.5,3.5,1,1), mgp = c(2, 0.6, 0), cex = 0.8, las = 1)
plot(intern)
```

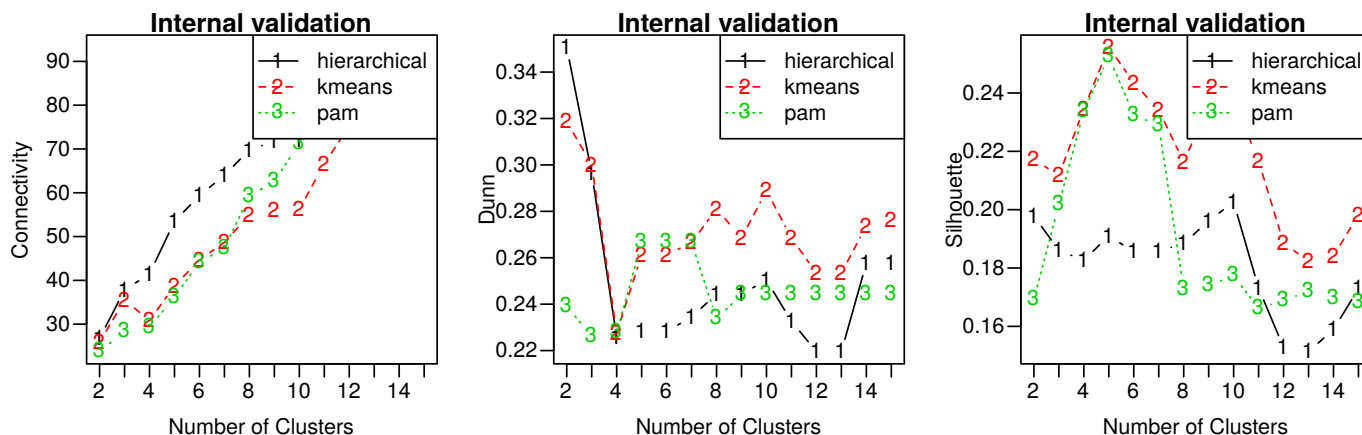


Figure 41:

Here is an example with more different clustering methods.

diana is a divisive hierarchical method, **fanny** is c-means fuzzy clustering (but the algorithm did not work here excepted for 2 clusters), **model** is model based clustering (based on gaussian models here) and **agnes** is equivalent to **hclust** with **ward.d2** method here while **hierarchical** is **hclust** with **ward.D** grouping method here. **pam** is the k-medoids algorithm and **kmeans** the k-means algorithm.

If you look at the silhouette width, almost all methods seem to “agree” on a 5 clusters solution and all seem to do a good job excepted **hclust** with **ward.D** method (and model based clustering). Remember however that the 5 clusters solution of a non hierarchical algorithm like k-means might be quite different than the 5 clusters solution of a hierarchical algorithm like **hclust**.

Even if the **hclust** with **ward.D2** results (“**agnes**”) are slightly lower, the easiness of the method and the possibility to draw dendrograms and heatmaps to help the interpretation might be a big advantage in favor of this method.

However if you want to have well separated clusters, the Dunn index and Connectivity index tells you another story : you might prefer a solution with 2 clusters and **agnes** i.e. **ward.D2** hierarchical clustering (remember that the Connectivity index must be minimized while the Dunn index must be maximized like the Silhouette width index).

Again the **ward.D2** hierarchical clustering approach seems to be an interesting compromise here. You can draw a dendrogram and a heatmap like figure 28. If you want well separated clusters (with a higher distance between the border of the clusters) you can stop the interpretation at the 2 major groups. If the separatedness of the clusters is less important you might interpret the 5 groups (that have indeed here clear biological interpretation...).

NB : see `?clValid-class` for the arguments of the `plot` method and `?legend` for more arguments of the legend

```
intern <- clValid(hell_trans, 2:15,
                 clMethods = c("hierarchical", "kmeans", "diana", "fanny",
                              "model", "pam", "agnes"),
                 method = "ward", validation = "internal")
# dev.new(width = 11/2.54, height = 20/2.54)
par(mfrow = c(3,1))
par(mar = c(3,3,3,1), mgp = c(1.9, 0.5, 0), cex = 0.8, las = 1)
plot(intern, legendLoc="top", measures = c("Connectivity"), main = "",
     bty = "n", ncol = 4, xpd = NA, inset = -0.3) # legend options
```

```
par(mar = c(3,3,0.5,1), mgp = c(1.9, 0.5, 0), cex = 0.8, las = 1)
plot(intern, legend = FALSE, measures = c("Dunn", "Silhouette"), main = "", bty = "n")
```

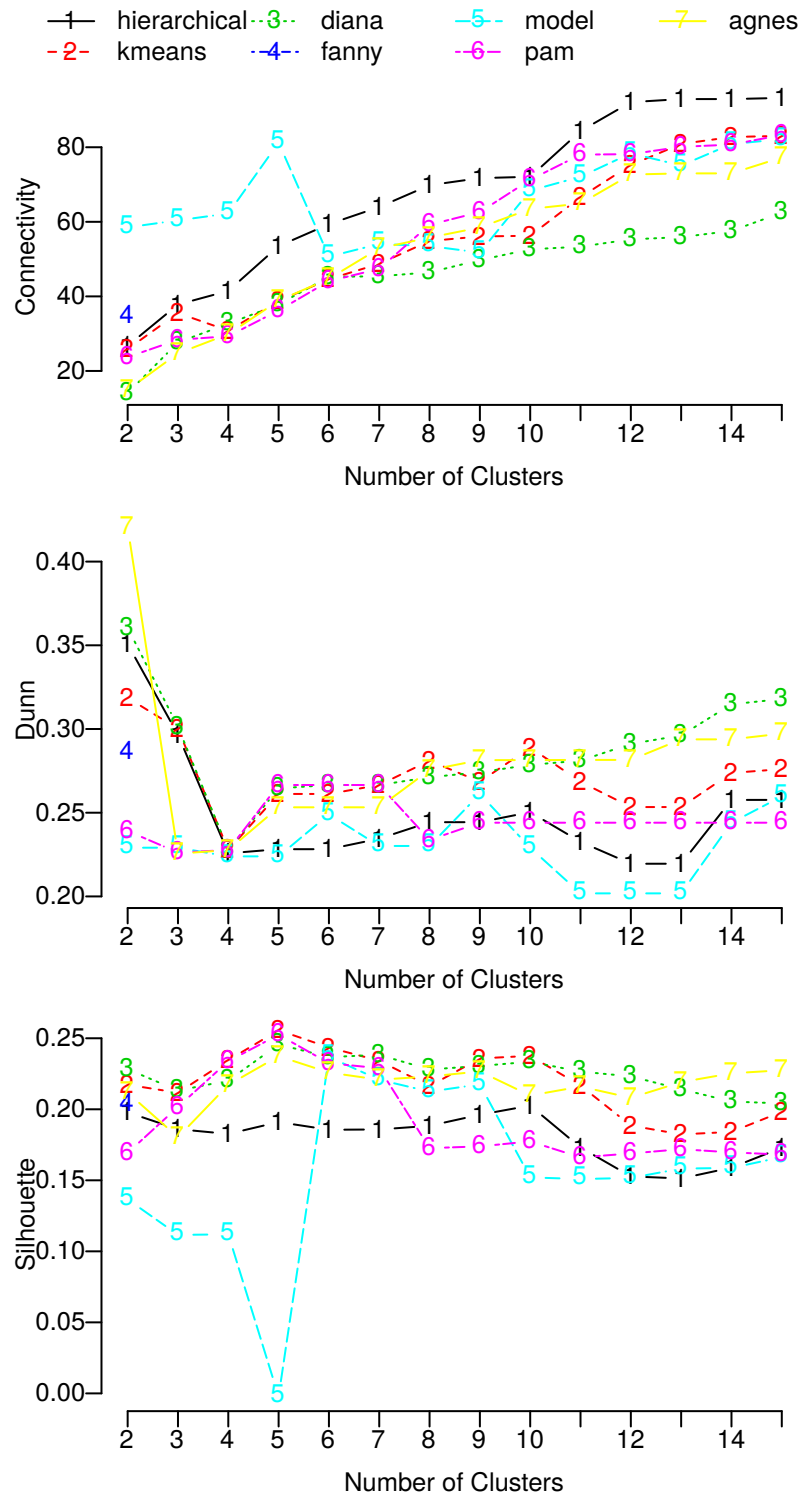


Figure 42: