Package 'constr.hclust'

July 19, 2022

Type Package

Version 1.6-2

Date 2022-07-19

Encoding UTF-8

Title Space- And Time-Constrained Clustering Package

Author Pierre Legendre and Guillaume Guénard

Maintainer Guillaume Guénard <guillaume.guenard@gmail.com>

Depends R (>= 3.5.0)

Imports sf, sp, spdep

Description Space-constrained or time-constrained agglomerative clustering from a multivariate dissimilarity matrix <doi:10.18637/jss.v103.i07>.

License GPL-3

LazyLoad yes

NeedsCompilation yes

RoxygenNote 7.2.0

R topics documented:

constr.hclust *Space- And Time-Constrained Clustering*

Description

Function constr.hclust carries out space-constrained or time-constrained agglomerative clustering from a multivariate dissimilarity matrix.

Usage

```
constr.hclust(
  d,
  method = "ward.D2",
  links,
  coords,
  beta = -0.25,
  chron = FALSE,
  members = NULL
)
```

Arguments

d	A dist-class dissimilarity (distance) matrix
method	The agglomeration method to be used (default: "ward.D2"; see details)
links	A list of edges (or links) connecting the points. May be omitted in some cases; see details and examples
coords	Coordinates of the observations (data rows) in the dissimilarity matrix d. The coordinates are used for plotting maps of the clustering results. This matrix may be omitted when the user does not wish to print maps of the clustering results or when no links file is provided. coords is a matrix or data frame with two columns, following the convention of the Cartesian plane: first column for abscissa, second column for ordinate. See examples
beta	The beta parameter for beta-flexible clustering (default: beta = -0.25)
chron	Logical (TRUE or FALSE) indicating whether a chronological (i.e. time-constrained or spatial transect) clustering should be calculated (default: chron = FALSE)
members	NULL or a vector with length size of d (default: NULL; See details)

Details

The agglomeration method to be used should be (an unambiguous abbreviation of) one of "ward.D", "ward.D2", "single", "complete", "average" (UPGMA), "mcquitty" (WPGMA), "centroid" (UPGMC), "median" (WPGMC), or "flexible". Method "ward.D2" (default) implements the Ward (1963) clustering criterion, method "ward.D" does not (Murtagh and Legendre, 2014).

Agglomerative clustering can be carried out with a constraint of spatial or temporal contiguity. This means that only the objects that are linked in links are considered to be candidates for clustering:

the next pair of objects to cluster will be the pair that has the lowest dissimilarity value among the pairs that are linked.

The same rule applies during the subsequent clustering steps, which involve groups of objects: the list of links is updated after each agglomeration step. All objects that are neighbours of one of the components that have fused are now neighbours of the newly formed cluster.

The edges (links) are specified using argument `links`, which can be an object of class `nb` (see, e.g., `tri2nb`), an object of class `listw` (see, e.g., `nb2listw`), a two-element `list` or an object coercible as a such (e.g., a two-column `dataframe`), or a two-column matrix with each row representing an edge and the columns representing the two ends of the edges. For lists with more than two elements, as well as dataframes or matrices with more than two-columns, only the first two elements or columns are used for the analysis. The edges are interpreted as being non directional; there is no need to specify an edge going from point a to point b and one going from point b to point a. While doing so is generally inconsequential for the analysis, it carries some penalty in terms of computation time. It is a good practice to place the nodes in increasing order of numbers from the top to the bottom and from the left to the right of the list but this is not mandatory. A word of caution: in cases where clusters with identical minimum distances occur, the order of the edges in the list may have an influence on the result. Alternative results would be statistically equivalent.

When argument `link` is omitted, regular (unconstrained) clustering is performed and a `hclust-class` object is returned unless argument `chron = TRUE`. When argument `chron = TRUE`, chronological clustering is performed, taking the order of observations as their positions in the sequence. Argument `links` is not used when `chron = TRUE`. Argument `chron` allows one to perform a chronological clustering in the case where observations are ordered chronologically. Here, the term "chronologically" should not be taken restrictively: the method remains applicable to other sequential data sets such as spatial series made of observations along a transect.

When the graph described by `link` is not entirely connected, a warning message is issued to warn the user about the presence and number of disjoint clusters and a procedure is suggested to identify the disjoint clusters. The disjoint clusters (or singletons) are merged in the order of their indices (i.e. the two clusters with smallest indices are merged first) and so on until all of disjoint clusters have been merged. The dissimilarity at which these clusters are merged is a missing value (`NA`) in vector `height` (i.e., unconnected clusters have undefined dissimilarities in constrained clustering).

If `members != NULL`, then `d` is taken to be a dissimilarity matrix between clusters instead of dissimilarities between individual objects. Then, `members` must be a vector giving the number of observations per cluster. In this way, the hierarchical clustering algorithm can be 'started in the middle of the dendrogram', e.g., in order to reconstruct the part of the tree above a cut. See examples in `hclust` for details on that functionality."

Memory storage and time to compute constrained clustering for N objects. The Lance and Williams algorithm for agglomerative clustering uses dissimilarity matrices. The amount of memory needed to store the dissimilarities among N observations as 64-bit double precision floating point variables (IEEE 754) is 8*N*(N-1)/2 bytes. For example, a dissimilarity matrix among 22 500 observations would require 2 024 910 000 bytes (1.89 GiB) of storage whereas one among 100 000 observations would take up 39 999 600 000 bytes (37.25 GiB). The implementation in this function needs to cache a copy of the dissimilarity matrix as its elements are modified following each merging of the closest clusters or singletons, thereby doubling the amounts of required memory shown above. Memory needed to store the other information associated with the clustering is much smaller. Users should make sure to have the necessary memory space (and system stability) before attempting to analyze large data sets. What is considered a large amount of memory has increased over time as computer hardware evolved with time. We let users apply contemporary common sense as to what

sample sizes represent manageable clustering problems. Computation time grows with N at roughly the same speed as memory storage requirement to store the dissimilarity matrices increases. See the Benchmarking example below.

With large data sets, a manageable output describing the classification of the sites is obtained with function cutree(x, k) where k is the number of groups. A dendrogram would be unreadable.

Value

A constr.hclust-class object.

Author(s)

Pierre Legendre <pierre.legendre@umontreal.ca> (preliminary version coded in R) and Guillaume Guénard <guillaume.guenard@umontreal.ca> (present version mostly coded in C)

References

Guénard, G. and P. Legendre. 2022. Hierarchical clustering with contiguity constraint in R. Journal of Statistical Software 103(7): 1-12 <doi:10.18637/jss.v103.i07>

Langfelder, P. and S. Horvath. 2012. Fast R functions for robust correlations and hierarchical clustering. Journal of Statistical Software 46: 1-17. https://www.jstatsoft.org/v46/i11/

Legendre, P. and L. Legendre. 2012. Numerical ecology, 3rd English edition. Elsevier Science BV, Amsterdam.

Murtagh, F. and P. Legendre. 2014. Ward's hierarchical agglomerative clustering method: which algorithms implement Ward's criterion? Journal of Classification 31: 274-295. doi: 10.1007/s00357-014-9161-z

Ward, J. H. 1963. Hierarchical grouping to optimize an objective function. Journal of the American Statistical Association 58: 236-244.

See Also

plot.constr.hclust, hclust, cutree, and ScotchWhiskey

Examples

```
##
### First example: Artificial map data from Legendre & Legendre
###                (2012, Fig. 13.26): n = 16
##
dat <- c(41,42,25,38,50,30,41,43,43,41,30,50,38,25,42,41)
coord.dat <- matrix(c(1,3,5,7,2,4,6,8,1,3,5,7,2,4,6,8,
                      4.4,4.4,4.4,4.4,3.3,3.3,3.3,3.3,
                      2.2,2.2,2.2,2.2,1.1,1.1,1.1,1.1),16,2)
##
### Obtaining a list of neighbours:
library(spdep)
listW <- nb2listw(tri2nb(coord.dat), style="B")
links.mat.dat <- listw2mat(listW)
neighbors <- listw2sn(listW)[,1:2]
```

```
##
### Calculating the (Euclidean) distance between points:
D.dat <- dist(dat)
##
### Display the points:
plot(coord.dat, type='n',asp=1)
title("Delaunay triangulation")
text(coord.dat, labels=as.character(as.matrix(dat)), pos=3)
for(i in 1:nrow(neighbors))
    lines(rbind(coord.dat[neighbors[i,1],],
          coord.dat[neighbors[i,2],]))
##
### Unconstrained clustring by hclust:
grpWD2_hclust <- hclust(D.dat, method="ward.D2")
plot(grpWD2_hclust, hang=-1)
##
### Clustering without a contiguity constraint;
### the result is represented as a dendrogram:
grpWD2_constr_hclust <- constr.hclust(D.dat, method="ward.D2")
plot(grpWD2_constr_hclust, hang=-1)
##
### Clustering with a contiguity constraint described by a list of
### links:
grpWD2cst_constr_hclust <-
    constr.hclust(
        D.dat, method="ward.D2",
        neighbors, coord.dat)
##
### To visualize using hclust's plotting method:
### stats:::plot.hclust(grpWD2cst_constr_hclust, hang=-1)
##
### Plot the results on a map with k=3 clusters:
plot(grpWD2cst_constr_hclust, k=3, links=TRUE, las=1, xlab="Eastings",
     ylab="Northings", cex=3, lwd=3)
##
### Generic functions from hclust can be used, for instance to obtain
### a list of members of each cluster:
cutree(grpWD2cst_constr_hclust, k=3)
##
### Now with k=5 clusters:
plot(grpWD2cst_constr_hclust, k=5, links=TRUE, las=1, xlab="Eastings",
     ylab="Northings", cex=3, lwd=3)
cutree(grpWD2cst_constr_hclust, k=5)
##
## End of the artificial map example

##
### Second example: Scotch Whiskey distilleries clustered using tasting
### scores (nose, body, palate, finish, and the four distances combined)
### constrained with respect to the distillery locations in Scotland.
##
## Documentation file about the Scotch Whiskey data: ?ScotchWhiskey
##
```

```
data(ScotchWhiskey)
### Cluster analyses for the nose, body, palate, and finish D
### matrices:
grpWD2cst_ScotchWhiskey <-
    lapply(
        ScotchWhiskey$dist,    ## A list of distance matrices
        constr.hclust,         ## The function called by function lapply
        links=ScotchWhiskey$neighbors@data,         ## The list of links
        coords=ScotchWhiskey$geo@coords/1000
    )
##
### The four D matrices (nose, body, palate, finish), represented as
### vectors in the ScotchWiskey data file, are combined as follows to
### produce a single distance matrix integrating all four types of
### tastes:
Dmat <- ScotchWhiskey$dist
ScotchWhiskey[["norm"]] <-
    sqrt(Dmat$nose^2 + Dmat$body^2 + Dmat$palate^2 + Dmat$finish^2)
##
### This example shows how to apply const.clust to a single D matrix when
### the data file contains several matrices.
grpWD2cst_ScotchWhiskey[["norm"]] <-
    constr.hclust(
        d=ScotchWhiskey[["norm"]],method="ward.D2",
        ScotchWhiskey$neighbors@data,
        coords=ScotchWhiskey$geo@coords/1000
    )
##
### A fonction to plot the Whiskey clustering results
plotWhiskey <- function(wh, k) {
   par(fig=c(0,1,0,1))
   plot(grpWD2cst_ScotchWhiskey[[wh]], k=k, links=TRUE, las=1,
        xlab="Eastings (km)", ylab="Northings (km)", cex=0.1, lwd=3,
        main=sprintf("Feature: %s",wh))
   text(ScotchWhiskey$geo@coords/1000,labels=1:length(ScotchWhiskey$geo))
   legend(x=375, y=700, lty=1L, lwd=3, col=rainbow(1.2*k)[1L:k],
          legend=sprintf("Group %d",1:k), cex=1.25)
   SpeyZoom <- list(xlim=c(314.7,342.2), ylim=c(834.3,860.0))
   rect(xleft=SpeyZoom$xlim[1L], ybottom=SpeyZoom$ylim[1L],col="#E6E6E680",
        xright=SpeyZoom$xlim[2L], ytop=SpeyZoom$ylim[2L], lwd=2, lty=1L)
   par(fig=c(0.01,0.50,0.46,0.99), new=TRUE)
   plot(grpWD2cst_ScotchWhiskey[[wh]], xlim=SpeyZoom$xlim,
        ylim=SpeyZoom$ylim, k=k, links=TRUE, las=1, xlab="", ylab="",
        cex=0.1, lwd=3, axes=FALSE)
   text(ScotchWhiskey$geo@coords/1000,labels=1:length(ScotchWhiskey$geo))
   rect(xleft=SpeyZoom$xlim[1L], ybottom=SpeyZoom$ylim[1L],
        xright=SpeyZoom$xlim[2L], ytop=SpeyZoom$ylim[2L], lwd=2, lty=1L)
}
##
### Plot the clustering results on the map of Scotland for 5 groups.
### The inset map shows the Speyside distilleries in detail:
plotWhiskey("nose", 5L)
plotWhiskey("body", 5L)
```

```
plotWhiskey("palate", 5L)
plotWhiskey("finish", 5L)
plotWhiskey("norm", 5L)
##
## End of the Scotch Whiskey tasting data example

## Not run:
##
### Third example: Fish community composition along the Doubs River,
### France. The sequence is analyzed as a case of chronological
### clustering, substituting space for time.
##
library(ade4)
library(adespatial)
data(doubs, package="ade4")
Doubs.D <- dist.ldc(doubs$fish, method="hellinger")
grpWD2cst_fish <- constr.hclust(Doubs.D, method="ward.D2", chron=TRUE,
                                coords=as.matrix(doubs$xy))
plot(grpWD2cst_fish, k=5, las=1, xlab="Eastings (km)",
     ylab="Northings (km)", cex=3, lwd=3)
##
### Repeat the plot with other values of k (number of groups)
##
## End of the Doubs River fish assemblages example

##
### Example with 6 connected points, shown in Fig. 2 of Guénard & Legendre paper
##
var = c(1.5, 0.2, 5.1, 3.0, 2.1, 1.4)
ex.Y = data.frame(var)
##
## Site coordinates, matrix xy
x.coo = c(-1, -2, -0.5, 0.5, 2, 1)
y.coo = c(-2, -1, 0, 0, 1, 2)
ex.xy = data.frame(x.coo, y.coo)
##
## Matrix of connecting edges E
from = c(1,1,2,3,4,3,4)
to = c(2,3,3,4,5,6,6)
ex.E = data.frame(from, to)
##
## Carry out constrained clustering analysis
test.out <-
    constr.hclust(
        dist(ex.Y),       # Response dissimilarity matrix
        method="ward.D2", # Clustering method
        links=ex.E,       # File of link edges (constraint) E
        coords=ex.xy      # File of geographic coordinates
    )
##
par(mfrow=c(1,2))
## Plot the map of the results for k = 3
plot(test.out, k=3)
```

```
## Plot the dendrogram
stats:::plot.hclust(test.out, hang=-1)
##

### Same example modified: disjoint clusters
##  Same ex.Y and ex.xy as in the previous example
var = c(1.5, 0.2, 5.1, 3.0, 2.1, 1.4)
ex.Y = data.frame(var)
##
## Site coordinates, matrix xy
x.coo = c(-1, -2, -0.5, 0.5, 2, 1)
y.coo = c(-2, -1, 0, 0, 1, 2)
ex.xy = data.frame(x.coo, y.coo)
##
## Matrix of connecting edges E2
from = c(1,1,2,4,4)
to = c(2,3,3,5,6)
ex.E2 = data.frame(from, to)
##
## Carry out constrained clustering analysis
test.out2 <-
    constr.hclust(
        dist(ex.Y),       # Response dissimilarity matrix
        method="ward.D2", # Clustering method
        links=ex.E2,      # File of link edges (constraint) E
        coords=ex.xy      # File of geographic coordinates
    )
cutree(test.out2, k=2)
##
par(mfrow=c(1,2))
## Plot the map of the results for k = 3
plot(test.out2, k=3)
## Plot the dendrogram showing the disconnected groups
stats:::plot.hclust(test.out2, hang=-1)
axis(2,at=0:ceiling(max(test.out2$height,na.rm=TRUE)))
##
## End of the disjoint clusters example
##
### Benchmarking example
### Benchmarking can be used to estimate computation time for different
### values of N.
### Computing time grows with N at roughly the same speed as the memory
### storage requirements to store the dissimilarity matrices.
##
require(magrittr)
require(pryr)
##
benchmark <- function(nobj) {
    # Argument -
    # nobj : Number of objects in simulation runs
    res <- matrix(NA,length(nobj),3) %>% as.data.frame
    colnames(res) <- c("N.objects","Storage (MiB)","Time (sec)")
    res[,1L] <- nobj
```

```
    ## resources <- list()
    for(i in 1:length(nobj)) {
        N <- nobj[i]
        coords.mem <- cbind(x=runif(N,-1,1),y=runif(N,-1,1))
        dat.mem <- runif(N,0,1)
        if(i>1L) rm(D.mem) ; gc()
        D.mem <- try(dat.mem %>% dist)  #; gc()
        if(any(class(D.mem)=="try-error"))
            break
        neighbors.mem <-
            (coords.mem %>%
                 tri2nb %>%
                 nb2listw(style="B") %>%
                 listw2sn)[,1:2]
        {start.time = Sys.time()
            res.mem <- try(constr.hclust(D.mem, method="ward.D2",
                                         neighbors.mem))
            end.time = Sys.time()}
        if(any(class(res.mem)=="try-error"))
            break
        res[i,2L] <- (2*object_size(D.mem) + object_size(neighbors.mem) +
                          object_size(res.mem))/1048576  # n. bytes per MiB
        res[i,3L] <- end.time-start.time
    }
    res[["N.objects"]] <- as.integer(res[["N.objects"]])
    res
}
res <- benchmark(nobj=c(1000,2000,5000,10000,20000,50000,100000))
##
### Plotting the results:
ok <- res %>% apply(1L, function(x) !x %>% is.na %>% any)
par(mar=c(3,6,2,2),mfrow=c(2L,1L))
barplot(height = res[ok,"Time (sec)"], names.arg= res[ok,"N.objects"],
        ylab="Time (seconds)\n",xlab="",las=1L,log="y")
par(mar=c(5,6,0,2))
barplot(height = res[ok,"Storage (MiB)"], names.arg= res[ok,"N.objects"],
        ylab="Total storage (MB)\n",xlab="Number of observations",
        las=1L,log="y")
##
### Examine the output file
res
##
## End of the benchmarking example

## End(Not run)
### End of examples
```

`constr.hclust-class` *Class For Constrained Hiereachical Clustering*

Description

Files belonging to this class hold information about the constrained agglomerative clustering and allows one to display results graphically.

Format

A file belonging to this class is a list with elements:

merge A (n-1) by 2 matrix. Row i of file `"merge"` describes the merging of clusters at step i of the clustering. If an element j in the row is negative, it means that observation -j was merged at this stage. If j is positive, it means that the merge was with the cluster formed at the (earlier) stage j of the algorithm. Thus negative entries in file `"merge"` indicate agglomerations of singletons, and positive entries indicate agglomerations of non-singletons.

height A set of (n-1) non-decreasing real values. The clustering height is the value of the criterion associated with the clustering method for the particular agglomeration.

order A vector giving the permutation of the original observations suitable for plotting, in the sense that a cluster plot using this ordering and matrix merge will not have crossing branches.

labels Labels for the clustered objects.

method The agglomerative clustering method that has been used.

call The call that produced the result.

dist.method The distance that has been used to create dissimilarity matrix `"d"` (only returned if the dissimilarity matrix object has a "method" attribute attached to it).

links A copy of the list of edges (if a matrix of edges was provided to the function).

coords A copy of the coordinates (if coordinates were provided to the function).

Details

The class inherits from `hclust-class` and describes the tree produced by the constrained clustering procedure.

All class members except `links` and `coords` are identical to those in `hclust-class`. several methods designed to process these objects are expected to also work with `constr.hclust-class` objects.

See Also

`hclust-class`

`constr.lshclust` *Space- And Time-Constrained Least Squares Clustering (Experimental)*

Description

Function `constr.lshclust` carries out space-constrained or time-constrained agglomerative clustering from a data matrix.

Usage

```
constr.lshclust(x, links, coords, chron = FALSE, output = "RSS")
```

Arguments

`x`	A data matrix
`links`	A list of edges (or links) connecting the points. May be omitted in some cases; see details and examples
`coords`	Coordinates of the observations (data rows) in matrix `d`. The coordinates are used for plotting maps of the clustering results; may be omitted. A matrix or data frame with two columns, following the convention of the Cartesian plane: first column for abscissa, second column for ordinates. See examples
`chron`	Logical (TRUE or FALSE) indicating whether a chronological (i.e. time-constrained) clustering should be calculated (default: `chron = FALSE`)
`output`	The type of edge lengths to return: square root sums of squares (`"RSS"`, the default), sums of squares (`"SS"`), Euclidean distance between centroids (`"D"`), square Euclidean distance between centroids (`"D2"`)

Details

Agglomerative clustering can be carried out with a constraint of spatial or temporal contiguity. This means that only the objects that are linked in `links` are considered to be candidates for clustering: the next pair of objects to cluster will be the pair that has the lowest dissimilarity value among the pairs that are linked.

The same rule applies during the subsequent clustering steps, which involve groups of objects: the list of links is updated after each agglomeration step. All objects that are neighbours of one of the components that have fused are now neighbours of the newly formed cluster.

The edges (links) are specified using argument `links`, which can be an object of class `nb` (see, e.g., `tri2nb`), an object of class `listw` (see, e.g., `nb2listw`), a two-element `list` or an object coercible as a such (e.g., a two-column `dataframe`), or a two-column matrix with each row representing an edge and the columns representing the two ends of the edges. For lists with more than two elements, as well as dataframes or matrices with more than two-columns only the first two elements or columns are used for the analysis. The edges are interpreted as being non directional; there is no need to specify an edge going from point a to point b and one going from point b to point a. While doing so is generally inconsequential for the analysis, it carries some penalty in terms of

computation time. It is a good practice to place the nodes in increasing order of numbers from the top to the bottom and from the left to the right of the list but this is not mandatory. A word of caution: in cases where clusters with identical minimum distances occur, the order of the edges in the list may have an influence on the result. Alternative results would be statistically equivalent.

When argument `link` is omitted, regular (unconstrained) clustering is performed and a `hclust-class` object is returned unless argument `chron = TRUE`. When argument `chron = TRUE`, chronological clustering is performed, taking the order of observations as their positions in the sequence. Argument `links` is not used when `chron = TRUE`. Argument `chron` allows one to perform a chronological clustering in the case where observations are ordered chronologically. Here, the term "chronologically" should not be taken restrictively: the method remains applicable to other sequential data sets such as spatial series made of observations along a transect.

When the graph described by `link` is not entirely connected, the resulting disjoint clusters (or singletons) are merged in the order of their indices (ie. the two clusters with smallest indices are merged until all of them have been merged), the dissimilarity at which these clusters are merged (`$height`) is assumed to be a missing value (`NA`), and a warning message is issued to warn the user about the presence and number of disjoint clusters.

Memory storage and time to compute constrained clustering for N objects. — The least squares clustering procedure generally uses less computer memory as does the Lance and Williams algorithm implemented by function `constr.hclust` because it does not use dissimilarity matrices. Internally, the function makes to copies of data matrix `x`, one that is used to accumulate the values as the clusters are being formed and one that is squared and used to accumulate the squared values during the clustering process. The amount of memory needed to store accumulation arrays for N observations described by M variables as 64-bit double precision floating point variables (IEEE 754) is the 8*N*M*2 bytes. It scales linearly with increasing sample size. By contrast, the dissimilarity matrix used by the Lance and Williams algorithm needs 8*N*(N-1)/2 bytes of storage; as much storage as when M = (N-1)/4. Since M is much smaller than (N-1)/4 for many practical cases, especially those when N is very large (e.g., many hundred thousands or millions), performing least squares hierarchical clustering will be faster (See the Benchmarking example below), require less storage, and be applicable to cases with larger N than the distance-based Lance and Williams algorithm, but at the price of a single classificatory criterion (i.e., within-cluster least squares).

With large data sets, a manageable output describing the classification of the sites is obtained with function `cutree`(x, k) where k is the number of groups.

Value

A `constr.hclust-class` object.

Author(s)

Pierre Legendre <pierre.legendre@umontreal.ca> and Guillaume Guénard <guillaume.guenard@umontreal.ca>

References

Guénard, G. and P. Legendre. 2022. Hierarchical clustering with contiguity constraint in R. Journal of Statistical Software 103(7): 1-12 <doi:10.18637/jss.v103.i07>

Legendre, P. and L. Legendre. 2012. Numerical ecology, 3rd English edition. Elsevier Science BV, Amsterdam.

See Also

plot.constr.hclust, hclust, and cutree

Examples

```
##
### First example: Artificial map data from Legendre & Legendre
###                (2012, Fig. 13.26): n = 16
##
dat <- c(41,42,25,38,50,30,41,43,43,41,30,50,38,25,42,41)
coord.dat <- matrix(c(1,3,5,7,2,4,6,8,1,3,5,7,2,4,6,8,
                      4.4,4.4,4.4,4.4,3.3,3.3,3.3,3.3,
                      2.2,2.2,2.2,2.2,1.1,1.1,1.1,1.1),16,2)

##
### Obtaining a list of neighbours:
library(spdep)
listW <- nb2listw(tri2nb(coord.dat), style="B")
links.mat.dat <- listw2mat(listW)
neighbors <- listw2sn(listW)[,1:2]
##
### Display the points:
plot(coord.dat, type='n',asp=1)
title("Delaunay triangulation")
text(coord.dat, labels=as.character(as.matrix(dat)), pos=3)
for(i in 1:nrow(neighbors))
    lines(rbind(coord.dat[neighbors[i,1],],
                coord.dat[neighbors[i,2],]))
##
### Clustering without a contiguity constraint;
### the result is represented as a dendrogram:
grpWD2_constr_lshclust <- constr.lshclust(x=dat, output="RSS")
plot(grpWD2_constr_lshclust, hang=-1)
##
### Clustering with a contiguity constraint described by a list of
### links:
grpWD2cst_constr_lshclust <-
    constr.lshclust(
        dat, neighbors,
        coord.dat, output="RSS")
##
### Plot the results on a map with k=3 clusters:
plot(grpWD2cst_constr_lshclust, k=3, links=TRUE, las=1, xlab="Eastings",
     ylab="Northings", cex=3, lwd=3)
##
### Generic functions from hclust can be used, for instance to obtain
### a list of members of each cluster:
cutree(grpWD2cst_constr_lshclust, k=3)
##
### Now with k=5 clusters:
plot(grpWD2cst_constr_lshclust, k=5, links=TRUE, las=1, xlab="Eastings",
     ylab="Northings", cex=3, lwd=3)
```

```
cutree(grpWD2cst_constr_lshclust, k=5)
##
## End of the artificial map example

##
### Third example: Scotch Whiskey distilleries clustered using four tasting
### scores (nose, body, palate, and finish) constrained with respect to the
### distillery locations in Scotland.
##
## Documentation file about the Scotch Whiskey data: ?ScotchWhiskey
##
data(ScotchWhiskey)
### Cluster analyses for the colour, nose, body, palate, and finish using
### least squares on the basis of the Mahalanobis metric.
##
### Combining the data matrices:
cols <-
    contr.treatment(
        n = nlevels(ScotchWhiskey$colour)
    )[ScotchWhiskey$colour,]
dimnames(cols) <- list(
    rownames(ScotchWhiskey$geo[,"Distillery"]),
    levels(ScotchWhiskey$colour)[-1L]
)
WhiskeyDat <- cbind(
    cols,
    ScotchWhiskey$body,
    ScotchWhiskey$palate,
    ScotchWhiskey$finish
)
rm(cols)
##
### Transforming WhiskeyDat into an orthonormal matrix using the Cholesky
### factorization: the least squares will relate to the Mahalanobis metric.
WhiskeyTr <- WhiskeyDat %*% solve(chol(cov(WhiskeyDat)))
grpWD2cst_ScotchWhiskey <-
    constr.lshclust(
        x=WhiskeyTr,
        links=ScotchWhiskey$neighbors@data,
        coords=ScotchWhiskey$geo@coords/1000
    )
##
### A fonction to plot the Whiskey clustering results
plotWhiskey <- function(k) {
    par(fig=c(0,1,0,1))
    plot(grpWD2cst_ScotchWhiskey, k=k, links=TRUE, las=1,
         xlab="Eastings (km)", ylab="Northings (km)", cex=0.1, lwd=3)
    text(ScotchWhiskey$geo@coords/1000,labels=1:length(ScotchWhiskey$geo))
    legend(x=375, y=700, lty=1L, lwd=3, col=rainbow(1.2*k)[1L:k],
           legend=sprintf("Group %d",1:k), cex=1.25)
    SpeyZoom <- list(xlim=c(314.7,342.2), ylim=c(834.3,860.0))
    rect(xleft=SpeyZoom$xlim[1L], ybottom=SpeyZoom$ylim[1L], col="#E6E6E680",
         xright=SpeyZoom$xlim[2L], ytop=SpeyZoom$ylim[2L], lwd=2, lty=1L)
```

```
    par(fig=c(0.01,0.50,0.46,0.99), new=TRUE)
    plot(grpWD2cst_ScotchWhiskey, xlim=SpeyZoom$xlim,
         ylim=SpeyZoom$ylim, k=k, links=TRUE, las=1, xlab="", ylab="",
         cex=0.1, lwd=3, axes=FALSE)
    text(ScotchWhiskey$geo@coords/1000,labels=1:length(ScotchWhiskey$geo))
    rect(xleft=SpeyZoom$xlim[1L], ybottom=SpeyZoom$ylim[1L],
         xright=SpeyZoom$xlim[2L], ytop=SpeyZoom$ylim[2L], lwd=2, lty=1L)
}
##
### Plot the clustering results on the map of Scotland for 5 groups.
### The inset map shows the Speyside distilleries in detail:
plotWhiskey(k=5L)
##
## End of the Scotch Whiskey tasting data example
##
## Not run:
##
### Third example: Fish community composition along the Doubs River,
### France. The sequence is analyzed as a case of chronological
### clustering, substituting space for time.
##
library(ade4)
data(doubs, package="ade4")
##
### Using the Hellinger metric on the species abundances:
Doubs.hel <- sqrt(doubs$fish / rowSums(doubs$fish))
Doubs.hel[rowSums(doubs$fish)==0,] <- 0
grpWD2cst_fish <- constr.lshclust(x=Doubs.hel, chron=TRUE,
                                  coords=as.matrix(doubs$xy))
##
plot(grpWD2cst_fish, k=5, las=1, xlab="Eastings (km)",
     ylab="Northings (km)", cex=3, lwd=3)
##
### Repeat the plot with other values of k (number of groups)
##
## End of the Doubs River fish assemblages example

##
### Benchmarking example
### Benchmarking can be used to estimate computation time for different
### values of N (number of sites) and M (number of variables).
##
require(magrittr)
require(pryr)
##
benchmark <- function(N, M) {
    res <- matrix(NA,length(N)*length(M),4L) %>% as.data.frame
    colnames(res) <- c("N(obs)","M(var)","Storage (MiB)","Time (sec)")
    res[[1L]] <- rep(N,length(M))
    res[[2L]] <- rep(M,each=length(N))
    ##
    for(i in 1:nrow(res)) {
        ## i=1L
```

```
        cat("N:",res[i,1L]," M:",res[i,2L],"\n")
        coords.mem <- cbind(x=runif(res[i,1L],-1,1),y=runif(res[i,1L],-1,1))
        if(i>1L) rm(dat.mem) ; gc()
        dat.mem <- try(matrix(runif(res[i,1L]*res[i,2L],0,1),
                              res[i,1L],res[i,2L]))
        if(any(class(dat.mem)=="try-error"))
            break
        neighbors.mem <-
            (coords.mem %>%
             tri2nb %>%
             nb2listw(style="B") %>%
             listw2sn)[,1:2]
        {start.time = Sys.time()
         res.mem <- try(constr.lshclust(dat.mem, neighbors.mem))
         end.time = Sys.time()}
        if(any(class(res.mem)=="try-error"))
            break
        res[i,3L] <- (3*object_size(dat.mem) + object_size(neighbors.mem) +
                      object_size(res.mem))/1048576  # n. bytes per MiB
        res[i,4L] <- as.numeric(end.time) - as.numeric(start.time)
    }
    res[["N(obs)"]] <- as.integer(res[["N(obs)"]])
    res[["M(var)"]] <- as.integer(res[["M(var)"]])
    res
}
##
N <- c(1000,2000,5000,10000,20000,50000)
M <- c(1,2,5,10,20,50)
res <- benchmark(N, M)
##
##
### Plotting the results:
par(mar=c(3,6,2,2),mfrow=c(2L,1L))
barplot(height = matrix(res[,"Time (sec)"],length(N),length(M)),
        names.arg = N, ylab = "Time (seconds)\n", xlab = "",
        las = 1L, log = "y", beside=TRUE)
par(mar=c(5,6,0,2))
barplot(height = matrix(res[,"Storage (MiB)"],length(N),length(M)),
        names.arg = N, ylab = "Total storage (MB)\n",
        xlab = "Number of observations", las = 1L, log = "y", beside=TRUE)
##
### Examine the output file
res
##
### Analyze how computing time and storage scales up with increasing number
### of observations and variables.
lm(log(`Time (sec)`)~log(`N(obs)`)+log(`M(var)`), data=res)
lm(log(`Storage (MiB)`)~log(`N(obs)`)+log(`M(var)`), data=res)

## End(Not run)
##
### End of the benchmarking example
##
```

```
### End of examples
##
```

`Faithful` *Old Faithful Erruption Interval Data Set*

Description

Erruption intervals during a time period of approximately eleven years

Usage

```
data(Faithful)
```

Format

A two-column data frame:

Interval Mean time interval, in hours, between two consecutive erruptions.

Date Median POSIX time of the averaging window (see below).

Details

Erruption timing data of Old Faithful geyser (Yellowstone National Park, WY, USA) between 2000 and 2011 were downloaded from website http://www.geyserstudy.org (access time: Sun Dec 13 14:43:25 2020). The original data series had 58527 eruption times and showed discontinuities because of interruptions in data recording (caused by equipment malfunction, maintenance, or onboard computer memory overrun caused by harsh weather conditions interferring with satelite data transmission. The series was thus split into 20 segments with continuous observations and the intervals between the eruptions were calculated for each of them.

For exemplary purposes, the segments were decimated to a smaller size by averaging individual time intervals using a 50-eruption wide Hann window and then the decimated data segments were concatenated into a single data series of 1142 estimates of the mean eruption interval.

Source

Guillaume Guenard <guillaume.guenard@gmail.com> and, Pierre Legendre <pierre.legendre@umontreal.ca>; http://www.geyserstudy.org

See Also

related data set `faithful` in `datasets-package` and `geyser` from package `MASS`.

Examples

```
data(Faithful)

## Distance matrix (Euclidean) of the mean intervals:

dst <- dist(Faithful$Interval)

## Segmenting the series with respect to eruption time intervals using
## chronological clustering (time-constrained Lance-Williams hierarchical
## agglomerative clustering):

chcl <- constr.hclust(dst, coords = Faithful$Date, chron = TRUE)

## Plotting the results:

### Partition sizes and the colors to display them:
parts <- c(2,3,4,5,6,7)
cols <- list(
  c("red","purple"),
  c("red","blue","purple"),
  c("red","orange","blue","purple"),
  c("red","orange","yellow","blue","purple"),
  c("red","orange","yellow","green","blue","purple"),
  c("red","orange","yellow","green","aquamarine","blue","purple")
)

### Plotting partitions with 2-7 segments:
par(mar=c(5,6.5,2,2))
plot(x=range(Faithful$Date), y=c(0.5,6.5), type="n", xlab="Time",
     ylab="Partitions\n\n", yaxt="n")
for(i in 1L:length(parts)) {
  chcl$coords[,"y"] <- i
  plot(chcl, parts[i], link=TRUE, lwd=25, hybrids="none",
       lwd.pt=0.5, cex=0, pch=21, plot=FALSE, lend=2, col=cols[[i]])
}
axis(2, at=1:length(parts), labels=sprintf("%d groups",parts), las=1)

tmp <- which(!!diff(cutree(chcl,parts[i])))
data.frame(From=Faithful$Date[tmp],
           To=Faithful$Date[tmp+1])
```

`Oribates` *Borcard's Obitatid Mite Data Set*

Description

Oribatid mite community data in a peat bog surrounding Lac Geai, QC, Canada

Usage

```
data(Oribates)
```

Format

A list with six elements:

fau A data frame with 70 rows (sites) and 35 columns (species) whose contents are the abundances of the species in the sites.

env A data frame with 70 rows (sites) and five columns (variables) whose contents are environmental variables taken on the sites.

xy Cartesian coordinates of the sites in the study area.

link A list of edges between neighboring locations (see details).

topo A list of color values for representing the topography of the study area.

map A raw color raster of the topography of the study area.

Details

Variables of `oribatid$env` are:

SubsDens Substrate density (g/L).

WatrCont Water content of the peat (g/L)

Substrate A seven-level factor describing the substrate (more on that subject below.

Shrub A three-level factor describing the presence and abundance of shrubs (mainly Ericaceae) on the peat surface.

topo A two-level factor describing the microtopography of the peat mat.

Levels of `oribatid$env$Substrate` are described as follows:

Sphagn1 Sphagnum magellanicum (with a majority of S. rubellum).

Sphagn2 Sphagnum rubellum.

Sphagn3 Sphagnum nemoreum (with a minority of S. angustifolium).

Sphagn4 Sphagnum rubellum and S. magellanicum in equal parts.

Litter Ligneous litter.

Barepeat Bare peat.

Interface Interface between Sphagnum species.

Levels of `oribatid$env$Shrub` where: "none", "few", and "many" (the variable may also be considered semi-quantitative), whereas levels of `oribatid$env$topo` were "Blanket" (ie. flat) and "Hummock" (ie. raised).

`Oribates$map` is a color raster generated from Fig. 1 in Borcard et al. 1994. It has dimensions 244 (number of pixels along the Y axis) by 940 (number of pixels along the X axis) and describes an area of 2.6m (Y axis) by 10m (W axis) with a resolution of approximately 10.6mm per pixel. A higher resolution image from the same data can also be found as Fig. 1.1 in Borcard et al. 2018 (see references below). The X axis corresponds to locations going from the edge of the water to the edge of the forest. The Y axis correspond the distances along the lake's shore.

Author(s)

Daniel Borcard, <daniel.borcard@umontreal.ca> and Pierre Legendre <pierre.legendre@umontreal.ca>

References

Borcard, D. and Legendre, P. 1994. Environmental Control and Spatial Structure in Ecological Communities: An Example Using Oribatid Mites (Acari, Oribatei). Environ. Ecol. Stat. 1(1): 37–61

Borcard, D., Legendre, P., and Drapeau, P. 1992. Partialling out the spatial component of ecological variation. Ecology, 73, 1045–1055.

Borcard, D.; Legendre, P.; and Gillet, F. 2018. Numerical Ecology with R (2nd Edition) Sprigner, Cham, Switzerland. ISBN 978-3-319-71403-5

See Also

Data set `oribatid` from package `ade4`, which is another version of this data set.

Examples

```
data("Oribates",package="constr.hclust")

## A map of the study area with the links.
par(mar=rep(0,4L))
plot(NA,xlim=c(0,12),ylim=c(-0.1,2.5),yaxs="i",asp=1,axes=FALSE)
rasterImage(Oribates$map, 0, -0.1, 10, 2.5, interpolate=FALSE)
arrows(x0=0.15,x1=1.15,y0=0.1,y1=0.1,code=3,length=0.05,angle=90,lwd=2)
text(x=0.65,y=0.025,labels="1m")
invisible(
  apply(Oribates$link,1L,
        function(x,xy,labels) {
          segments(x0=xy[x[1L],1L],x1=xy[x[2L],1L],
                   y0=xy[x[1L],2L],y1=xy[x[2L],2L])
        },xy=Oribates$xy,labels=FALSE)
)
points(Oribates$xy,cex=1.25,pch=21,bg="black")
legend(10.1,2.5,legend=Oribates$topo[["Type"]],pt.bg=Oribates$topo[["RGB"]],
       pch=22L,pt.cex=2.5)

## Hellinger distance on the species composition matrix.
Oribates.hel <- dist(sqrt(Oribates$fau/rowSums(Oribates$fau)))

## Constrained clustering of the sites on the basis of their species
## composition.
Oribates.chclust <- constr.hclust(d=Oribates.hel, links=Oribates$link,
                                  coords=Oribates$xy)

## Plotting with different numbers of clusters.
par(mfrow=c(4,1),mar=c(0.5,0,0.5,0))
cols <- c("turquoise", "orange", "blue", "violet", "green", "red", "purple")
parts <- c(2,3,5,7)
for(i in 1L:length(parts)) {
```

```
  plot(NA, xlim=c(0,10), ylim=c(-0.1,2.5), xaxs="i", yaxs="i", asp=1,
       axes=FALSE)
  arrows(x0=0.15, x1=1.15, y0=0.1, y1=0.1, code=3, length=0.05, angle=90,
         lwd=2)
  text(x=0.65, y=0, labels="1m", cex=1.5)
  plot(Oribates.chclust, parts[i], links=TRUE, plot=FALSE,
       col=cols[round(seq(1,length(cols),length.out=parts[i]))], lwd=4,
       cex=2.5, pch=21, hybrids="single", lwd.hyb=0.25, lty.hyb=3)
  text(x=0.25, y=2.25, labels=LETTERS[i], cex=2.5)
}
```

`plot.constr.hclust` *Plotting Method For Space- And Time-Constrained Clustering*

Description

Method `plot.constr.hclust` displays the results of space-constrained or time-constrained agglomerative cluster analyses obtained from multivariate dissimilarity matrices.

Usage

```
## S3 method for class 'constr.hclust'
plot(x, k, xlim, ylim, xlab, ylab, links,
points=TRUE, pch=21L, hybrids=c("change","single","none"), lty.hyb=1L,
lwd.hyb=1, col.hyb="black", plot=TRUE, col, axes, cex=1, lty, lwd, lwd.pt=1,
invert.axes=FALSE, ...)
```

Arguments

`x`	A `constr.hclust-class` object
`k`	The number of clusters to delineate
`xlim`	Limits, in abscissa, of the zone to be plotted
`ylim`	Limits, in ordinate, of the zone to be plotted
`xlab`	Labels for x axis annotation
`ylab`	Labels for y axis annotation
`links`	Should segments be drawn to represent the edges (links) (default: FALSE)
`points`	Should observation points be drawn (default: TRUE)
`pch`	Point character to display observations (default: 21, a circle with a background color)
`hybrids`	How should hybrid segments be drawn (default: "change")
`lty.hyb`	Line type to use for hybrid segments (default: lty)
`lwd.hyb`	Width of hybrid segments with respect to lwd (default: 1)
`col.hyb`	Colour of hybrid segments, when applicable (default: "black")

plot	Should a new plotting window be opened first (default: TRUE)
col	Colours to use for the k different clusters (see details). Default: col=rainbow)
axes	Should the axes be displayed (default: TRUE)
cex	Text and symbol magnification (see graphical parameters) (default: 1)
lty	Reference line type (see graphical parameters for details)
lwd	Reference line width (see graphical parameters for details)
lwd.pt	Line width around points with respect to lwd (default: 1)
invert.axes	Should axes be inverted on the plot (default: FALSE)
...	Other graphical parameters

Details

The plotting method uses the coordinates provided by the user of constr.hclust to display the observations. It cuts the tree (see cutree) into k clusters and uses the colours provided by the user as argument col to display each cluster using the indices returned by cutree. When links = TRUE, each edge is displayed as a segments with colours corresponding to the clusters at its two ends. A special treatment is done for hybrids edges: those whose ends lie in different clusters; it is controlled by argument hybrids. When argument hybrids="change" (the default), hybrid links are represented as segments whose colours change halfway. When hybrids="single", hybrid edges are shown as single-color lines, whose color is given as argument col.hyb, whereas hybrids="none" suppresses the drawing of hybrid edges. Whenever hybrid edges are displayed, their width with respect to the lwd value is controlled by argument lwd.hyb.

When argument plot=FALSE, no plot command is issued and the points (and segments when links = TRUE) are drawn over an existing plotting window. This functionality is to allow one to plot the result of a constrained clustering over an existing map. In that case, arguments xlim, ylim, axes, and all other graphical parameters to which the method plot would responds are ignored.

The default colours are generated by function rainbow; see palette for further details on using colour palettes in R. The colour palette can be changed by the user.

When disjoint clusters are present (i.e., when the graph provided to constr.hclust is not entirely connected), the function does not allow one to plot fewer clusters than the number of disjoint subsets; a warning message is issued to notify the user.

Author(s)

Guillaume Guénard <guillaume.guenard@umontreal.ca> and Pierre Legendre <pierre.legendre@umontreal.ca>

Examples

```
##
### Artificial map data from Legendre & Legendre (2012, Fig. 13.26)
### n = 16
##
dat <- c(41,42,25,38,50,30,41,43,43,41,30,50,38,25,42,41)
coord.dat <- matrix(c(1,3,5,7,2,4,6,8,1,3,5,7,2,4,6,8,
                      4.4,4.4,4.4,4.4,3.3,3.3,3.3,3.3,
                      2.2,2.2,2.2,2.2,1.1,1.1,1.1,1.1),16,2)
```

```
##
### Obtaining a list of neighbours:
library(spdep)
listW <- nb2listw(tri2nb(coord.dat), style="B")
links.mat.dat <- listw2mat(listW)
neighbors <- listw2sn(listW)[,1:2]
##
### Calculating the (Euclidean) distance between points:
D.dat <- dist(dat)
##
### Display the points:
plot(coord.dat, type='n',asp=1)
title("Delaunay triangulation")
text(coord.dat, labels=as.character(as.matrix(dat)), pos=3)
for(i in 1:nrow(neighbors))
    lines(rbind(coord.dat[neighbors[i,1],],
          coord.dat[neighbors[i,2],]))
##
### Clustering with a contiguity constraint described by a list of
### links:
grpWD2cst_constr_hclust <-
    constr.hclust(
        D.dat, method="ward.D2",
        neighbors, coord.dat)
##
### Plot the results with k=5 clusters on a map:
plot(grpWD2cst_constr_hclust, k=5, links=TRUE, las=1,
     xlab="Eastings", ylab="Northings", cex=3, lwd=3)
##
### Repeat the plot with other values of k (number of groups)
```

`ScotchWhiskey` *Scotch Whiskey Data Set*

Description

Single Malt Scotch whiskeys from 109 distilleries

Usage

```
data(ScotchWhiskey)
```

Format

A list with 12 members:

geo A `SpatialPointsDataFrame-class` object containing the geographic coordinates and other information about the distilleries.

colour The whiskey colour coded as a 14-level factor.

nose A set of 12 nasal notes (boolean).

body A set of 8 body notes (boolean).

palate A set of 15 palatine notes (boolean).

finish A set of 19 finish (or after-taste) notes (boolean).

nbChar Number of characteristics attributed to each distillery for each of the four sets of boolean features: nose, body, palate, finish.

listW A `listw` object (see `nb2listw`) containing information about the spatial edges (neighbour links) between the distilleries.

links.mat A binary square matrix of the spatial connexions between the distilleries (contiguity matrix).

neighbors A `SpatialLinesDataFrame-class` object containing geographic information about the spatial links between the distilleries.

dist A list of distance matrices obtained for each of the four sets of boolean features.

Details

There are 5 data sets: color, nose, body, palate, and finish. The binary (0,1) descriptors are in the same order as on p. 239 of the whisky paper.

There are two whiskies in the classification from the Springbank distillery. One pertains to the Islay group, the other to the Western group.

Please let us know of the analyses you have performed with the whiskey data, especially if you intend to publish them.

The distance matrices were calculated separately as follows for each tasting data set:

D = (1 - S4)^0.5,

where S4 is the Simple matching coefficient of Sokal & Michener (1958). This coefficient was called S4 in the Gower & Legendre (1986) paper and S1 in the Legendre & Legendre (2012) book. In package `ade4`, coefficient D = sqrt(1 - S4) is computed by function `dist.binary` using argument `"method=2"`.

Source

Pierre Legendre <pierre.legendre@umontreal.ca> and François-Joseph Lapointe <francois-joseph.lapointe@umontreal.ca>, Département de sciences biologiques, Université de Montréal, Montréal, Québec, Canada.

References

Lapointe, F.-J. and P. Legendre. 1994. A classification of pure malt Scotch whiskies. Applied Statistics 43: 237-257 <http://www.dcs.ed.ac.uk/home/jhb/whisky/lapointe/text.html>.

Gower, J.C. and Legendre, P. 1986. Metric and Euclidean properties of dissimilarity coefficients. Journal of Classification, 3, 5-48.

Legendre, P. and Legendre, L. 2012. Numerical Ecology. 3rd English edition. Elsevier Science BV, Amsterdam.

Examples

```
data(ScotchWhiskey)
lapply(ScotchWhiskey,ncol)
ScotchWhiskey$nbChar
ScotchWhiskey$listW  ## attr(ScotchWhiskey$listW,"class")
names(ScotchWhiskey)
names(ScotchWhiskey$dist)

plotWhiskey <- function(main) {
    plot(x=ScotchWhiskey$geo@coords[,1L]/1000,
         xlab="Eastings (km)",
         y=ScotchWhiskey$geo@coords[,2L]/1000,
         ylab="Northings (km)",
         main=main,
         type="n",asp=1)
    apply(
        ScotchWhiskey$neighbor@data,1L,
        function(X,coords) {
            segments(
                coords[X[1L],1L]/1000,
                coords[X[1L],2L]/1000,
                coords[X[2L],1L]/1000,
                coords[X[2L],2L]/1000
            )
        },
        coords=ScotchWhiskey$geo@coords
    )
    invisible(NULL)
}

plotWhiskey("Scotch whiskey: peat nose")
cols <- c("blue","orange")
points(ScotchWhiskey$geo@coords/1000,pch=21L,
       bg=cols[ScotchWhiskey$nose[,"peat"]+1L])
legend(x=50,y=1000,legend=c("Has a peat nose","Has no peat nose"),
       pch=21L,pt.bg=rev(cols))

plotWhiskey("Scotch whiskey: soft body")
cols <- c("red","green")
points(ScotchWhiskey$geo@coords/1000,pch=21L,
       bg=cols[ScotchWhiskey$body[,"soft"]+1L])
legend(x=50,y=1000,legend=c("Has a soft body","Has no soft body"),
       pch=21L,pt.bg=rev(cols))

plotWhiskey("Scotch whiskey: spicy palate")
cols <- c("red","green")
points(ScotchWhiskey$geo@coords/1000,pch=21L,
       bg=cols[ScotchWhiskey$palate[,"spice"]+1L])
legend(x=50,y=1000,legend=c("Has a spicy palate","Has no spicy palate"),
       pch=21L,pt.bg=rev(cols))

plotWhiskey("Scotch whiskey: sweet finish")
```

```
cols <- c("red","green")
points(ScotchWhiskey$geo@coords/1000,pch=21L,
       bg=cols[ScotchWhiskey$finish[,"sweet"]+1L])
legend(x=50,y=1000,legend=c("Has a sweet finish","Has no sweet finish"),
       pch=21L,pt.bg=rev(cols))

## To visualize (part of) the distance matrices:
as.matrix(ScotchWhiskey$dist$nose)[1:5,1:5]
as.matrix(ScotchWhiskey$dist$body)[1:5,1:5]
as.matrix(ScotchWhiskey$dist$palate)[1:5,1:5]
as.matrix(ScotchWhiskey$dist$finish)[1:5,1:5]

## The data tables:
ScotchWhiskey$colour
head(ScotchWhiskey$nose)
head(ScotchWhiskey$body)
head(ScotchWhiskey$palate)
head(ScotchWhiskey$finish)
```

Index