Package 'SOMbrero'

October 12, 2022

Title SOM Bound to Realize Euclidean and Relational Outputs

Version 1.4-1

Date 2022-01-03

Maintainer Nathalie Vialaneix <nathalie.vialaneix@inrae.fr>

Description The stochastic (also called on-line) version of the Self-Organising Map (SOM) algorithm is provided. Different versions of the algorithm are implemented, for numeric and relational data and for contingency tables as described, respectively, in Kohonen (2001) <isbn:3-540-67921-9>, Olteanu & Villa-Vialaneix (2005) <doi:10.1016/j.neucom.2013.11.047> and Cottrell et al (2004) <doi:10.1016/j.neunet.2004.07.010>. The package also contains many plotting features (to help the user interpret the results), can handle (and impute) missing values and is delivered with a graphical user interface based on 'shiny'.

BugReports https://github.com/tuxette/sombrero/issues

Depends R (>= 3.1.0), igraph (>= 1.0), markdown

Imports scatterplot3d, shiny, grDevices, graphics, stats, ggplot2, ggwordcloud, metR, interp

Suggests testthat, rmarkdown, knitr, hexbin, shinycssloaders, shinyBS, shinyjs, shinyjqui

License GPL (>= 2)

Repository CRAN

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.1.2

NeedsCompilation no

Author Nathalie Vialaneix [aut, cre],
Elise Maigne [aut],
Jerome Mariette [aut],
Madalina Olteanu [aut],
Fabrice Rossi [aut],

Laura Bendhaiba [ctb],
Julien Boelaert [ctb]

Date/Publication 2022-01-03 17:10:08 UTC

R topics documented:

`SOMbrero-package` *Self Organizing Maps Bound to Realize Euclidean and Relational Outputs*

Description

This package implements the stochastic (also called on-line) Self-Organizing Map (SOM) algorithms for numeric and relational data.

It is based on a grid (see `initGrid`), which is part of the parameters given to the algorithm (see `initSOM` and `trainSOM`). Many graphs can help you with the results (see `plot.somRes`).

Details

Package:	SOMbrero
Type:	Package
Version:	1.4-1
Date:	2022-01-03
License:	GPL (>= 2)

The version of the SOM algorithm implemented in this package is the stochastic version.

Several variants able to handle non-vectorial data are also implemented in their stochastic versions: `type = "korresp"` for contingency tables, as described in Cottrell et al. (2004) (with the observation weights defined in Cottrell and Letrémy, 2005a) and `type = "relational"` for dissimilarity data, as described in Olteanu and Villa-Vialaneix (2015a) with the fast implementation of Mariette *et al.* (2017). A special focus has been put on representing graphs, as described in Olteanu and Villa-Vialaneix (2015b).

In addition, the numeric version of the algorithm handles missing values: missing entries are not used during training but the resulting map can be used to fill missing entries (using the entry of the corresponding prototype). The method is taken from Cottrell and Letrémy (2005b).

Author(s)

Nathalie Vialaneix <nathalie.vialaneix@inrae.fr>
Élise Maigné <elise.maigne@inrae.fr>
Jérome Mariette <jerome.mariette@inrae.fr>
Madalina Olteanu <olteanu@ceremade.dauphine.fr>
Fabrice Rossi <fabrice.rossi@apiacoa.org>
Laura Bendhaïba <laurabendhaiba@gmail.com>
Julien Boelaert <julien.boelaert@gmail.com>

Maintainer: Nathalie Vialaneix <nathalie.vialaneix@inrae.fr>

References

Kohonen T. (2001) *Self-Organizing Maps*. Berlin/Heidelberg: Springer-Verlag, 3rd edition.

Cottrell M., Ibbou S., Letrémy P. (2004) SOM-based algorithms for qualitative variables. *Neural Networks*, **17**, 1149-1167.

Cottrell M., Letrémy P. (2005a) How to use the Kohonen algorithm to simultaneously analyse individuals in a survey. *Neurocomputing*, **21**, 119-138.

Cottrell M., Letrémy P. (2005b) Missing values: processing with the Kohonen algorithm. *Proceedings of Applied Stochastic Models and Data Analysis (ASMDA 2005)*, 489-496.

Letrémy P. (2005) Programmes basés sur l'algorithme de Kohonen et dediés à l'analyse des données. SAS/IML programs for 'korresp'. http://samm.univ-paris1.fr/Programmes-SAS-de-cartes-auto.

Mariette J., Rossi F., Olteanu M., Villa-Vialaneix N. (2017) Accelerating stochastic kernel SOM. In: M. Verleysen, *XXVth European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN 2017)*, i6doc, Bruges, Belgium, 269-274.

Olteanu M., Villa-Vialaneix N. (2015a) On-line relational and multiple relational SOM. *Neurocomputing*, **147**, 15-30.

Olteanu M., Villa-Vialaneix N. (2015b) Using SOMbrero for clustering and visualizing graphs. *Journal de la Société Française de Statistique*, **156**, 95-119.

Rossi F. (2013) yasomi: Yet Another Self-Organising Map Implementation. R package, version 0.3. https://github.com/fabrice-rossi/yasomi

Villa-Vialaneix N. (2017) Stochastic self-organizing map variants with the R package SOMbrero. In: J.C. Lamirel, M. Cottrell, M. Olteanu, *12th International Workshop on Self-Organizing Maps and Learning Vector Quantization, Clustering and Data Visualization (Proceedings of WSOM 2017)*, IEEE, Nancy, France.

See Also

initGrid, trainSOM, plot.somRes and sombreroGUI.

---

impute *Impute values from prototype information*

Description

Impute values by replacing missing entries with the corresponding assigned prototype entries

Usage

```
impute(object, ...)
```

Arguments

object	a somRes object.
...	unused.

Value

Imputed matrix as in Cottrell and Letrémy, (2005)

Author(s)

Nathalie Vialaneix <nathalie.vialaneix@inrae.fr>

References

Cottrell M., Letrémy P. (2005) Missing values: processing with the Kohonen algorithm. *Proceedings of Applied Stochastic Models and Data Analysis (ASMDA 2005)*, 489-496.

See Also

trainSOM

Examples

```
# Run trainSOM algorithm on the iris data with 500 iterations
set.seed(1505)
missings <- cbind(sample(1:150, 50, replace = TRUE),
                  sample(1:4, 50, replace = TRUE))
x.data <- as.matrix(iris[, 1:4])
x.data[missings] <- NA
iris.som <- trainSOM(x.data = x.data)
iris.som
impute(iris.som)
```

`initGrid` *Create an empty grid*

Description

Create an empty (square) grid equipped with topology.

Usage

```
initGrid(
  dimension = c(5, 5),
  topo = c("square", "hexagonal"),
  dist.type = c("euclidean", "maximum", "manhattan", "canberra", "minkowski",
    "letremy")
)
```

Arguments

`dimension`	a 2-dimensional vector giving the dimensions (width, length) of the grid
`topo`	topology of the grid. Accept values `"square"` (Default) or `"hexagonal"`.
`dist.type`	distance type that defines the topology of the grid (see 'Details'). Default to `"euclidean"`

Details

The units (neurons) of the grid are positionned at coordinates (1,1), (1,2), (1,3), ..., (2,1), (2,2), ..., for the `square` topology. The topology of the map is defined by a distance based on those coordinates, that can be one of `"euclidean"`, `"maximum"`, `"manhattan"`, `"canberra"`, `"minkowski"`, `"letremy"`, where the first 5 ones correspond to distance methods implemented in `dist` and `"letremy"` is the distance of the original implementation by Patrick Letrémy that switches between `"maximum"` and `"euclidean"` during the training.

Value

an object of class `myGrid` with the following entries:

- `coord` 2-column matrix with x and y coordinates of the grid units
- `topo` topology of the grid;
- `dim` dimensions of the grid (width corresponds to x coordinates)
- `dist.type` distance type that defines the topology of the grid.

Author(s)

Élise Maigné <elise.maigne@inrae.fr>
Madalina Olteanu <olteanu@ceremade.dauphine.fr>
Nathalie Vialaneix <nathalie.vialaneix@inrae.fr>

References

Letrémy P. (2005) Programmes basés sur l'algorithme de Kohonen et dédiés à l'analyse des données. SAS/IML programs for 'korresp'. http://samm.univ-paris1.fr/Programmes-SAS-de-cartes-auto

See Also

plot.myGrid for plotting the grid

Examples

```
initGrid()
initGrid(dimension=c(5, 7), dist.type = "maximum")
```

---

initSOM *Initialize parameters for the SOM algorithm*

Description

The initSOM function returns a paramSOM class object that contains the parameters needed to run the SOM algorithm.

Usage

```
initSOM(
  dimension = c(5, 5),
  topo = c("square", "hexagonal"),
  radius.type = c("gaussian", "letremy"),
  dist.type = switch(match.arg(radius.type), letremy = "letremy", gaussian =
    "euclidean"),
  type = c("numeric", "relational", "korresp"),
  mode = c("online"),
  affectation = c("standard", "heskes"),
  maxit = 500,
  nb.save = 0,
  verbose = FALSE,
  proto0 = NULL,
 init.proto = switch(type, numeric = "random", relational = "obs", korresp = "random"),
 scaling = switch(type, numeric = "unitvar", relational = "none", korresp = "chi2"),
  eps0 = 1
)

## S3 method for class 'paramSOM'
print(x, ...)

## S3 method for class 'paramSOM'
summary(object, ...)
```

Arguments

dimension	Vector of two integer points corresponding to the x dimension and the y dimension of the myGrid class object. Default values are: (5,5). Other data-driven defaults are set by function trainSOM.
topo	The topology to be used to build the grid of the myGrid class object. Accept values "square" (Default) or "hexagonal".
radius.type	The neighborhood type. Default value is "gaussian", which corresponds to a Gaussian neighborhood. The annealing of the neighborhood during the training step is similar to the one implemented in yasomi. The alternative value corresponds to an piecewise linear neighborhood as implemented by Patrick Letrémy in his SAS scripts.
dist.type	The neighborhood relationship on the grid. One of c("letremy", "euclidean", "maximum", "manhattan", "canberra", "minkowski"). When radius.type is letremy, default value is letremy which is the original implementation by Patrick Letrémy. When radius.type is gaussian, default value is euclidean. The other possible values are passed to method in function dist. dist.type = "letremy" is not permitted with radius.type = "gaussian". Only euclidian is allowed with hexagonal topology.
type	The SOM algorithm type. Possible values are: numeric (default value), korresp and relational.
mode	The SOM algorithm mode. Default value is online.
affectation	The SOM affectation type. Default value is standard which corresponds to a hard affectation. Alternative is heskes which corresponds to Heskes's soft affectation.
maxit	The maximum number of iterations to be done during the SOM algorithm process. Default value is 500. Other data-driven defaults are set by function trainSOM.
nb.save	The number of intermediate back-ups to be done during the algorithm process. Default value is 0.
verbose	The boolean value which activates the verbose mode during the SOM algorithm process. Default value is FALSE.
proto0	The initial prototypes. Default value is NULL.
init.proto	The method to be used to initialize the prototypes, which may be "random" (randomization), "obs" (each prototype is assigned a random observation) or "pca". In pca the prototypes are initialized to the observations closest to a grid along the two first principal components of the data (numeric case) or along a two-dimensional multidimensional scaling (relational case, equivalent to a relational PCA). Default value is random for the numeric and korresp types, and obs for the relational type. pca is not available for korresp SOM.
scaling	The type of data pre-processing. For numeric SOM, possibilities are unitvar (data are centered and scaled; this is the default value for a numeric SOM), none (no pre-processing), and center (data are centered but not scaled). For korresp SOM, the only available value is chi2. For relational SOM, possibilities are none (no pre-processing, default value for relational SOM) and cosine. This last one first turns the dissimilarity into a similarity using the suggestion

in (Lee and Verleysen, 2007). Then, a cosine normalization as described in (Ben-Hur and Weston, 2010) is applied to the kernel, that is finally turned back into its induced distance. For further details on this processing, have a look at the corresponding documentation in the directory "doc" of the package's installation directory.

`eps0` The scaling value for the stochastic gradient descent step in the prototypes' update. The scaling value for the stochastic gradient descent step is equal to $\frac{0.3\epsilon_0}{1+0.2t/\text{dim}}$ where t is the current step number and dim is the grid dimension (width multiplied by height).

`x` an object of class `paramSOM`.

`...` not used

`object` an object of class `paramSOM`.

Value

The `initSOM` function returns an object of class `paramSOM` which is a list of the parameters passed to the `initSOM` function, plus the default parameters for the ones not specified by the user.

Author(s)

Élise Maigné <elise.maigne@inrae.fr>
Madalina Olteanu <olteanu@ceremade.dauphine.fr>
Nathalie Vialaneix <nathalie.vialaneix@inrae.fr>

References

Ben-Hur A., Weston J. (2010) A user's guide to support vector machine. In: *Data Mining Techniques for the Life Sciences*, Springer-Verlag, 223-239.

Heskes T. (1999) Energy functions for self-organizing maps. In: *Kohonen Maps*, Oja E., Kaski S. (Eds.), Elsevier, 303-315.

Lee J., Verleysen M. (2007) *Nonlinear Dimensionality Reduction*. Information Science and Statistics series, Springer.

Letrémy P. (2005) Programmes basés sur l'algorithme de Kohonen et dediés à l'analyse des données. SAS/IML programs for 'korresp'. http://samm.univ-paris1.fr/Programmes-SAS-de-cartes-auto.

Rossi F. (2013) yasomi: Yet Another Self-Organising Map Implementation. R package, version 0.3. https://github.com/fabrice-rossi/yasomi

See Also

See `initGrid` for creating a SOM prior structure (grid).

Examples

```
# create a default 'paramSOM' class object
default.paramSOM <- initSOM()
summary(default.paramSOM)
```

`lesmis` *Dataset "Les Misérables"*

Description

This dataset contains the coappearance network (igraph object) of characters in the novel Les Misérables (written by the French writter Victor Hugo).

Format

`lesmis` is an `igraph` object. Its vertices are the characters of the novel and an edge indicates that the two characters appear together in the same chapter of the novel, at least once. Vertex attributes for this graph are '`id`', a vertex number between 1 and 77, and '`label`', the character's name. The edge attribute '`value`' gives the number of co-appearances between the two characters afferent to the edge (the `igraph` can thus be made a weighted graph using this attribute). Finally, a graph attribute '`layout`' is used to provide a layout (generated with the `igraph` function `layout_with_fr`) for visualizing the graph.

`dissim.lesmis` is a dissimilarity matrix computed with the function `shortest_paths` and containing the length of the shortest paths between pairs of nodes.

Details

Les Misérables is a French historical novel, written by Victor Hugo and published in 1862. The co-appearance network has been extracted by D.E. Knuth (1993).

References

Hugo V. (1862) *Les Miserables*.

Knuth D.E. (1993) *The Stanford GraphBase: A Platform for Combinatorial Computing*. Reading (MA): Addison-Wesley.

Examples

```
data(lesmis)
## Not run:
summary(lesmis)
plot(lesmis,vertex.size=0)
## End(Not run)
```

myGrid *Methods for 'myGrid' objects.*

Description

Methods for the result of `initGrid` (`myGrid` object)

Usage

```
## S3 method for class 'myGrid'
print(x, ...)

## S3 method for class 'myGrid'
summary(object, ...)

## S3 method for class 'myGrid'
plot(x, show.names = TRUE, names = 1:prod(x$dim), ...)
```

Arguments

`x`	`myGrid` object
`...`	Further arguments to the `plot` function.
`object`	`myGrid` object
`show.names`	Whether the cluster names must be printed in center of the grid or not. Default to `TRUE` (names not displayed).
`names`	If `show.names` = `TRUE`, values of the names to display. Default to the cluster number.

Details

The `myGrid` class has the following entries:

- `coord` 2-column matrix with x and y coordinates of the grid units
- `topo` topology of the grid;
- `dim` dimensions of the grid (width corresponds to x coordinates)
- `dist.type` distance type that defines the topology of the grid.

During plotting, the color filling process uses the coordinates of the object `x` included in `x$coord`.

Author(s)

Élise Maigné <elise.maigne@inrae.fr>
Madalina Olteanu, <olteanu@ceremade.dauphine.fr>
Nathalie Vialaneix, <nathalie.vialaneix@inrae.fr>

See Also

initGrid to define a myGrid class object.

Examples

```
# creating grid
a.grid <- initGrid(dimension=c(5,5), topo="square", dist.type="maximum")

# plotting grid
# without any color specification
plot(a.grid)
# generating colors from rainbow() function
my.colors <- grDevices::rainbow(5*5)
plot(a.grid) + ggplot2::scale_fill_manual(values = my.colors)
```

---

plot.somRes *Plot a* somRes *class object*

Description

Produce graphics to help interpreting a somRes object.

Usage

```
## S3 method for class 'somRes'
plot(
  x,
  what = c("obs", "prototypes", "energy", "add"),
  type = switch(what, obs = "hitmap", prototypes = "color", add = "pie", energy =
    "energy"),
  variable = NULL,
  my.palette = NULL,
  is.scaled = if (x$parameters$type == "numeric") TRUE else FALSE,
  show.names = TRUE,
 names = if (what != "energy") switch(type, graph = 1:prod(x$parameters$the.grid$dim),
    1:prod(x$parameters$the.grid$dim)) else NULL,
  proportional = TRUE,
  pie.graph = FALSE,
  pie.variable = NULL,
  s.radius = 1,
  view = if (x$parameters$type == "korresp") "r" else NULL,
  ...
)
```

Arguments

x	A somRes class object.
what	What you want to plot. Either the observations (obs, default case), the evolution of energy (energy), the prototypes (prototypes) or an additional variable (add).
type	Further argument indicating which type of chart you want to have. Choices depend on the value of what (what="energy" has no type argument). Default values are "hitmap" for obs, "color" for prototypes and "pie" for add. See section "Details" below for further details.
variable	Either the variable to be used for what="add" or the index of the variable of the data set to consider. For type="boxplot", the default value is the sequence from 1 to the minimum between 5 and the number of columns of the data set. In all other cases, default value is 1. See somRes.plotting for further details.
my.palette	A vector of colors. If omitted, predefined palettes are used, depending on the plot case. This argument is used for the following combinations: all "color" types and "prototypes"/"poly.dist".
is.scaled	A boolean indicating whether values should be scaled prior to plotting or not. Default value is TRUE when type="numeric" and FALSE in the other cases.
show.names	Boolean used to indicate whether each neuron should have a title or not, if relevant. Default to TRUE. It is feasible on the following cases: all "color", "lines", "meanline", "barplot", "boxplot", "names" types, "add"/"pie", "prototypes"/"umatrix", "prototypes"/"poly.dist" and "add"/"words".
names	The names to be printed for each neuron if show.names=TRUE. Default to a number which identifies the neuron.
proportional	Boolean used when what="add" and type="pie". It indicates if the pies should be proportional to the number of observations in the class. Default value is TRUE.
pie.graph	Boolean used when what="add" and type="graph". It indicates if the vertices should be pies or not.
pie.variable	The variable needed to plot the pies when what="add", type="graph" and argument pie.graph=TRUE.
s.radius	The size of the pies to be plotted (maximum size when proportional=TRUE) for what="add", type="graph" and pie.graph=TRUE. The default value is 0.9.
view	Used only when the algorithm's type is "korresp". It indicates whether rows ("r") or columns ("c") must be drawn.
...	Further arguments to be passed to the underlined plot function (which can be plot, barplot, pie... depending on type; see somRes.plotting for further details).

Details

See somRes.plotting for further details and more examples.

Author(s)

Élise Maigné <elise.maigne@inrae.fr>
Madalina Olteanu <olteanu@ceremade.dauphine.fr>
Nathalie Vialaneix <nathalie.vialaneix@inrae.fr>

See Also

trainSOM to run the SOM algorithm, that returns a somRes class object.

Examples

```
# run the SOM algorithm on the numerical data of 'iris' data set
iris.som <- trainSOM(x.data = iris[, 1:4], nb.save = 2)
# plots
# on energy
plot(iris.som, what = "energy")
# on observations
plot(iris.som, what = "obs", type = "lines")
# on prototypes
plot(iris.som, what = "prototypes", type = "3d", variable = "Sepal.Length")
# on an additional variable: the flower species
plot(iris.som, what = "add", type = "pie", variable = iris$Species)
```

---

predict.somRes *Predict the classification of a new observation*

Description

Predict the neuron where a new observation is classified

Usage

```
## S3 method for class 'somRes'
predict(object, x.new = NULL, ..., radius = 0)
```

Arguments

object	a somRes object.
x.new	a new observation (optional). Default values is NULL which corresponds to performing prediction on the training dataset.
...	not used.
radius	current radius used to perform soft affectation (when affectation = "heskes", see initSOM for further details about Heskes' soft affectation). Default value is '0', which corresponds to a hard affectation.

Details

The number of columns of the new observations (or its length if only one observation is provided) must match the number of columns of the data set given to the SOM algorithm (see `trainSOM`).

Value

`predict.somRes` returns the number of the neuron to which the new observation is assigned (i.e., neuron with the closest prototype).

When the algorithm's type is `"korresp"`, `x.new` must be the original contingency table passed to the algorithm.

Author(s)

Jérome Mariette <jerome.mariette@inrae.fr>
Madalina Olteanu <olteanu@ceremade.dauphine.fr>
Fabrice Rossi <fabrice.rossi@apiacoa.org>
Nathalie Vialaneix <nathalie.vialaneix@inrae.fr>

See Also

`trainSOM`

Examples

```
set.seed(2343)
my.som <- trainSOM(x.data = iris[-100, 1:4], dimension = c(5, 5))
predict(my.som, iris[100, 1:4])
```

---

presidentielles2002 *2002 French presidential election data set*

Description

This data set provides the number of votes at the first round of the 2002 French presidential election for each of the 16 candidates for 106 administrative districts called "Départements".

Format

`presidentielles2002` is a data frame of 106 rows (the French administrative districts called "Départements") and 16 columns (the candidates).

Source

The data are provided by the French ministry "Ministère de l'Intérieur". The original data can be downloaded at https://www.interieur.gouv.fr/Elections/Les-resultats/Presidentielles (2002 élections and "Résultats par départements").

References

The 2002 French presidential election consisted of two rounds. The second round attracted a greater than usual amount of international attention because of far-right candidate Le Pen's unexpected victory over Socialist candidate Lionel Jospin. The event is known because, on the one hand, the number of candidates was unusually high (16) and, on the other hand, because the polls had failed to predict that Jean-Marie Le Pen would be on the second round.

Further comments at https://en.wikipedia.org/wiki/2002_French_presidential_election.

Examples

```
data(presidentielles2002)
apply(presidentielles2002, 2, sum)
```

---

projectIGraph *Compute the projection of a graph on a grid*

Description

Compute the projection of a graph, provided as an igraph object, on the grid of the somRes object.

Usage

```
projectIGraph(object, init.graph, ...)
```

Arguments

object	a somRes object.
init.graph	an igraph whose number of vertices is equal to the clustering length of the somRes object.
...	Not used.

Value

The result is an igraph which vertexes are the clusters (the clustering is thus understood as a vertex clustering) and the edges are the counts of edges in the original graph between two vertices corresponding to the two clusters in the projected graph or, if init.graph is a weighted graph, the sum of the weights between the pairs of vertices corresponding to the two clusters.

The resulting igraph object's attributes are:

- the graph attribute layout which provides the layout of the projected graph according to the grid of the SOM;
- the vertex attributes name and size which, respectively are the vertex number on the grid and the number of vertexes included in the corresponding cluster;
- the edge attribute weight which gives the number of edges (or the sum of the weights) between the vertexes of the two corresponding clusters.

Author(s)

Madalina Olteanu <olteanu@ceremade.dauphine.fr>
Nathalie Vialaneix <nathalie.vialaneix@inrae.fr>

References

Olteanu M., Villa-Vialaneix N. (2015) Using SOMbrero for clustering and visualizing graphs. *Journal de la Société Française de Statistique*, **156**, 95-119.

See Also

`projectIGraph.somSC` which uses the results of a super-clustering to obtain another projected graph. `plot.somRes` with the option `type="graph"` or `plot.somSC` with the option `type="projgraph"`.

Examples

```
data(lesmis)
set.seed(7383)
mis.som <- trainSOM(x.data=dissim.lesmis, type="relational", nb.save=10)
proj.lesmis <- projectIGraph(mis.som, lesmis)
## Not run: plot(proj.lesmis)
```

---

protoDist *Compute distances between prototypes*

Description

Compute distances, either between all prototypes (`mode = "complete"`) or only between prototypes' neighbours (`mode = "neighbors"`).

Usage

```
protoDist(object, mode = c("complete", "neighbors"), radius = 1, ...)
```

Arguments

`object`	a `somRes` object.
`mode`	Specifies which distances should be computed (default to `"complete"`).
`radius`	Radius used to fetch the neighbors (default to 1). The distance used to compute the neighbors is the Euclidean distance.
`...`	Not used.

Details

When `mode="complete"`, distances between all prototypes are computed. When `mode="neighbors"`, distances are computed only between the prototypes and their neighbors. If the data were preprocessed during the SOM training procedure, the distances are computed on the normalized values of the prototypes.

Value

When mode = "complete", the function returns a square matrix which dimensions are equal to the product of the grid dimensions.

When mode = "neighbors", the function returns a list which length is equal to the product of the grid dimensions; the length of each item is equal to the number of neighbors. Neurons are considered to have 8 neighbors at most (*i.e.*, two neurons are neighbors if they have an Euclidean distance smaller than radius. Natural choice for radius is 1 for hexagonal topology and 1 or $\sqrt{2}$ for square topology (4 and 8 neighbors respectively).

Author(s)

Madalina Olteanu <olteanu@ceremade.dauphine.fr>
Nathalie Vialaneix <nathalie.vialaneix@inrae.fr>

See Also

trainSOM

Examples

```
set.seed(2343)
my.som <- trainSOM(x.data = iris[,1:4], dimension = c(5,5))
protoDist(my.som)
```

---

quality *Compute SOM algorithm quality criteria*

Description

The quality function computes several quality criteria for the result of a SOM algorithm.

Usage

```
quality(sommap, quality.type, ...)
```

Arguments

sommap	A somRes object (see trainSOM for details).
quality.type	The quality type to compute. Two types are implemented: quantization and topographic. The output of the function is one of those or both of them using the option "all". Default value is the latter.
...	Not used.

Value

The `quality` function returns either a numeric value (if only one type is computed) or a list a numeric values (if all types are computed).

The quantization error calculates the mean squared euclidean distance between the sample vectors and their respective cluster prototypes. It is a decreasing function of the size of the map.

The topographic error is the simplest of the topology preservation measure: it calculates the ratio of sample vectors for which the second best matching unit is not in the direct neighborhood of the best matching unit.

Author(s)

Madalina Olteanu <olteanu@ceremade.dauphine.fr>
Nathalie Vialaneix <nathalie.vialaneix@inrae.fr>

References

Polzlbauer G. (2004) Survey and comparison of quality measures for self-organizing maps. In: *Proceedings of the Fifth Workshop on Data Analysis (WDA'04), Paralic, J., Polzlbauer, G., Rauber, A. (eds)* Sliezsky dom, Vysoke Tatry, Slovakia: Elfa Academic Press, 67-82.

See Also

`trainSOM`, `plot.somRes`

Examples

```
my.som <- trainSOM(x.data = iris[,1:4])
quality(my.som, quality.type = "all")
quality(my.som, quality.type = "topographic")
```

`sombreroGUI` *Graphical Web User Interface for SOMbrero*

Description

Start the SOMbrero GUI.

Usage

```
sombreroGUI()
```

Value

This function starts the graphical user interface with the default system browser. This interface is more lickely to work properly with Firefox https://www.mozilla.org/fr/firefox/new/. In case Firefox is not your default browser, copy/paste http://localhost:8100 into the URL bar.

Author(s)

Élise Maigné <elise.maigne@inrae.fr>
Julien Boelaert <julien.boelaert@gmail.com>
Madalina Olteanu <olteanu@ceremade.dauphine.fr>
Nathalie Vialaneix <nathalie.vialaneix@inrae.fr>

References

Villa-Vialaneix N. (2017) Stochastic self-organizing map variants with the R package SOMbrero. In: J.C. Lamirel, M. Cottrell, M. Olteanu, *12th International Workshop on Self-Organizing Maps and Learning Vector Quantization, Clustering and Data Visualization (Proceedings of WSOM 2017)*, IEEE, Nancy, France.

RStudio and Inc. (2013). shiny: Web Application Framework for R. R package version 0.7.0. https://cran.r-project.org/package=shiny

---

`somRes.plotting` *Plotting* `somRes` *results*

Description

Useful details on how to produce graphics to help interpreting a `somRes` object.

Important: the graphics availables for the different types of SOM are marked with a N, a K or a R. (**N** = numerical SOM, **K** = korresp SOM and **R** = relational SOM).

Graphics on the observations: `what = "obs"`

For the cases `what = "obs"` and `what = "add"`, if a neuron is empty, nothing will be plotted at its location.

The possible values for `type` are:

`"hitmap"` **(K, R)** plots proportional areas according to the number of observations per neuron. It is the default plot when `what="obs"`.

`"color"` **(N)** can have one more argument, `variable`, the name or index of the variable to be considered (default, `1`, the first variable). Neurons are filled using the given colors according to the average value level of the observations for the chosen variable.

`"lines"` **(N)** plots a line for each observation in every neuron, between variables. A vector of variables (names or indexes) can be provided with the argument `variable`.

`"meanline"` **(N)** plots, for each neuron, the average value level of the observations, with lines and points. One point represents a variable. By default, all variables of the dataset used to train the algorithm are plotted but a vector of variables (names or indexes) can be provided with the argument `variable`.

`"barplot"` **(N)** is similar to `"meanline"` but using barplots. Then, a bar represents a variable.

"boxplot" **(N)** plots boxplots for the observations in every neuron, by variable. Like "lines", "meanline" and "barplot" a vector of variables (names or indexes) can be provided with the argument variable.

"names" **(N, K, R)** prints on the grid the element names (i.e., the row names or row and column names in the case of korresp) in the neuron to which it belongs.

Graphic on the energy: what = "energy" **(N, K, R)**

This graphic is only available if some intermediate backups have been registered (i.e., with the argument nb.save of trainSOM or initSOM resulting in x$parameters$nb.save>1). Graphic plots the evolution of the level of the energy according to the registered steps.

Graphics on the prototypes: what = "prototypes"

The possible values for type are:

"lines" **(N, K, R)** has the same behavior as the "lines" case described in the observations section, but according to the prototypes level.

"barplot" **(N, K, R)** has the same behavior as the "barplot" case described in the observations section, but according to the prototypes level.

"color" **(N, K)** has the same behavior as the "color" case described in the observations section, but according to the prototypes level.

"3d" **(N)** case is similar to the "color" case, but in 3 dimensions, with x and y the coordinates of the grid and z the value of the prototypes for the considered variable. This function can take two more arguments: maxsize (default to 2) and minsize (default to 0.5) for the size of the points representing neurons.

"smooth.dist" **(N, K, R)** depicts the average distance between a prototypes and its neighbors on a map where x and y are the coordinates of the prototypes on the grid.

"poly.dist" **(N, K, R)** also represents the distances between prototypes but with polygons plotted for each neuron. The closest from the border the polygon point is, the closest the pairs of prototypes are. The color used for filling the polygon shows the number of observations in each neuron. A white polygon means that there is no observation. With the default colors, a red polygon means a high number of observations.

"umatrix" **(N, K, R)** is another way of plotting distances between prototypes. The grid is plotted and filled with my.palette colors according to the mean distance between the current neuron and the neighboring neurons. With the default colors, red indicates proximity.

"mds" **(N, K, R)** plots the number of the neuron on a map according to a Multi Dimensional Scaling (MDS) projection on a two dimensional space.

"grid.dist" **(N, K, R)** plots on a 2 dimension map all distances. The number of points on this picture is equal to $\frac{\text{number of neurons} \times (\text{number of neurons} - 1)}{2}$. On the x axis corresponds to the prototype distances whereas the y axis depicts the grid distances.

Graphics on an additional variable: what="add"

The case what="add" considers an additional variable, which has to be given to the argument variable. Its length must match the number of observations in the original data.

When the algorithm's type is korresp, no graphic is available for what = "add".

The possible values for type are:

"color" **(N, R)** has the same behavior as the "color" case described in the observations section. Then, the additional variable must be a numerical vector.

"lines" **(N, R)** has the same behavior as the "lines" case described in the observations section. Then, the additional variable must be a numerical matrix or a data frame.

"boxplot" **(N, R)** has the same behavior as the "boxplot" case described in the observations section. Then, the additional variable must be either a numeric vector or a numeric matrix/data frame.

"barplot" **(N, R)** has the same behavior as the "barplot" case described in the observations section. Then, the additional variable must be either a numeric vector or a numeric matrix/data frame.

"pie" **(N)** requires the argument variable to be a vector, which will be passed to the function as.factor, and plots one pie for each neuron according to this factor. By default, the size of the pie is proportional to the number of observations affected to its neuron but this can be changed with the argument proportional = FALSE.

"names" **(N, R)** has the same behavior as the "names" case described in the observations section. Then, the names to be printed are the elements of the variable given to the variable argument. This case can take one more argument: size (default to 4) for the size of the words.

"words" **(N, R)** needs the argument variable be a numeric matrix or a data.frame: names of the columns will be used as words and the values express the frequency of a given word in the observation. Then, for each neuron of the grid, the words will be printed with sizes proportional to the sum of their values in the neuron. If the variable given is a contingency table, it will plot directly the frequency of the words in the neurons.

"graph" **(N, R)** requires that the argument variable is an igraph object (see library("igraph"). According to the existing edges in the graph and to the clustering obtained with the SOM algorithm, a clustered graph will be produced where a vertex between two vertices represents a neuron and the width of an edge is proportional to the number of edges in the given graph between the vertices affected to the corresponding neurons. The option can handle two more arguments: pie.graph and pie.variable. These are used to display the vertex as pie charts. For this case, pie.graph must be set to TRUE and a factor vector is supplied by pie.variable.

Further arguments via ...

Further arguments, their reference functions and the plot.somRes cases are summarized in the following list:

- plot.igraph is called by the cases:
  - what = "add" / type = "graph"
  - what = "add" / type = "projgraph" (for a superclass object)
- persp is called by the case what = "prototypes" / type = "3d"
- ggplot is called in all the other cases.

In complement to ggplot,

- geom_text_wordcloud is called by the cases:
  - type = "names"
  - what = "add" / type = "words"
- geom_contour_fill is called by the case what = "prototypes" / type = "smooth.dist"

Author(s)

Élise Maigné <elise.maigne@inrae.fr>
Madalina Olteanu <madalina.olteanu@univ-paris1.fr>
Nathalie Vialaneix <nathalie.vialaneix@inra.fr>

Examples

```
### Numerical SOM
# run the SOM algorithm on the numerical data of 'iris' data set
iris.som <- trainSOM(x.data = iris[,1:4], nb.save = 2)

####### energy plot
plot(iris.som, what = "energy") # energy

####### plots on observations
plot(iris.som, what = "obs", type = "hitmap")
## Not run:
plot(iris.som, what = "obs", type = "lines")
plot(iris.som, what = "obs", type = "barplot")
plot(iris.som, what = "obs", type = "boxplot")
plot(iris.som, what = "obs", type = "meanline")
plot(iris.som, what = "obs", type = "color", variable = 1)
plot(iris.som, what = "obs", type = "names")
## End(Not run)

####### plots on prototypes
plot(iris.som, what = "prototypes", type = "3d", variable = "Sepal.Length")
## Not run:
plot(iris.som, what = "prototypes", type = "lines")
plot(iris.som, what = "prototypes", type = "barplot")
plot(iris.som, what = "prototypes", type = "umatrix")
plot(iris.som, what = "prototypes", type = "color", variable = "Petal.Length")
plot(iris.som, what = "prototypes", type = "smooth.dist")
plot(iris.som, what = "prototypes", type = "poly.dist")
plot(iris.som, what = "prototypes", type = "grid.dist")
plot(iris.som, what = "prototypes", type = "mds")
## End(Not run)


####### plots on an additional variable: the flower species
plot(iris.som, what = "add", type = "pie", variable = iris$Species)
## Not run:
plot(iris.som, what = "add", type = "names", variable = iris$Species)
plot(iris.som, what = "add", type = "words", variable = iris[,1:2])
## End(Not run)
```

superClass *Create super-clusters from SOM results*

Description

Aggregate the resulting clustering of the SOM algorithm into super-clusters.

Usage

```
superClass(sommap, method, members, k, h, ...)

## S3 method for class 'somSC'
print(x, ...)

## S3 method for class 'somSC'
summary(object, ...)

## S3 method for class 'somSC'
plot(
  x,
  what = c("obs", "prototypes", "add"),
 type = c("dendrogram", "grid", "hitmap", "lines", "meanline", "barplot", "boxplot",
    "mds", "color", "poly.dist", "pie", "graph", "dendro3d", "projgraph"),
  plot.var = TRUE,
  show.names = TRUE,
  names = 1:prod(x$som$parameters$the.grid$dim),
  ...
)

## S3 method for class 'somSC'
projectIGraph(object, init.graph, ...)
```

Arguments

sommap	A somRes object.
method	Argument passed to the hclust function.
members	Argument passed to the hclust function.
k	Argument passed to the cutree function (number of super-clusters to cut the dendrogram).
h	Argument passed to the cutree function (height where to cut the dendrogram).
...	Used for plot.somSC: further arguments passed either to the function plot (case type="dendro") or to plot.myGrid (case type="grid") or to plot.somRes (all other cases).
x	A somSC object.
object	A somSC object.

what — What you want to plot for superClass object. Either the observations (obs), the prototypes (prototypes) or an additional variable (add), or NULL if not appropriate. Automatically set for types "hitmap" (to "obs"), 'grid' (to "prototypes"), default to "obs" otherwise. If what='add', the function plot.somRes will be called with the argument what set to "add".

type — The type of plot to draw. Default value is "dendrogram", to plot the dendrogram of the clustering. Case "grid" plots the grid in color according to the super clustering. Case "projgraph" uses an igraph object passed to the argument variable and plots the projected graph as defined by the function projectIGraph.somSC. All other cases are those available in the function plot.somRes and surimpose the super-clusters over these plots.

plot.var — A boolean indicating whether a graph showing the evolution of the explained variance should be plotted. This argument is only used when type="dendrogram", its default value is TRUE.

show.names — Whether the cluster titles must be printed in center of the grid or not for type="grid". Default to FALSE (titles not displayed).

names — If show.names = TRUE, values of the title to display for type="grid". Default to "Cluster " followed by the cluster number.

init.graph — An igraph object which is projected according to the super-clusters. The number of vertices of init.graph must be equal to the number of rows in the original dataset processed by the SOM (case "korresp" is not handled by this function). In the projected graph, the vertices are positionned at the center of gravity of the super-clusters (more details in the section **Details** below).

Details

The superClass function can be used in 2 ways:

- to choose the number of super clusters via an hclust object: then, both arguments k and h are not filled.
- to cut the clustering into super clusters: then, either argument k or argument h must be filled. See cutree for details on these arguments.

The squared distance between prototypes is passed to the algorithm.

summary on a superClass object produces a complete summary of the results that displays the number of clusters and super-clusters, the clustering itself and performs ANOVA analyses. For type="numeric" the ANOVA is performed for each input variable and test the difference of this variable accross the super-clusters of the map. For type="relational" a dissimilarity ANOVA is performed (see (Anderson, 2001), except that in the present version, a crude estimate of the p-value is used which is based on the Fisher distribution and not on a permutation test.

On plots, the different super classes are identified in the following ways:

- either with different color, when type is set among: "grid" (N, K, R), "hitmap" (N, K, R), "lines" (N, K, R), "barplot" (N, K, R), "boxplot", "poly.dist" (N, K, R), "mds" (N, K, R), "dendro3d" (N, K, R), "graph" (R), "projgraph" (R)
- or with title, when type is set among: "color" (N, K), "pie" (N, R)

In the list above, the charts available for a numerical SOM are maked with a N, with a K for a korresp SOM and with a R for relational SOM.

projectIGraph.somSC produces a projected graph from the igraph object passed to the argument variable as described in (Olteanu and Villa-Vialaneix, 2015). The attributes of this graph are the same than the ones obtained from the SOM map itself in the function projectIGraph.somRes. plot.somSC used with type="projgraph" calculates this graph and represents it by positionning the super-vertexes at the center of gravity of the super-clusters. This feature can be combined with pie.graph=TRUE to super-impose the information from an external factor related to the individuals in the original dataset (or, equivalently, to the vertexes of the graph).

Value

The superClass function returns an object of class somSC which is a list of the following elements:

- clusterThe super clustering of the prototypes (only if either k or h are given by user).
- treeAn hclust object.
- somThe somRes object given as argument (see trainSOM for details).

The projectIGraph.somSC function returns an object of class igraph with the following attributes:

- the graph attribute layout which provides the layout of the projected graph according to the center of gravity of the super-clusters positionned on the SOM grid;
- the vertex attributes name and size which, respectively are the vertex number on the grid and the number of vertexes included in the corresponding cluster;
- the edge attribute weight which gives the number of edges (or the sum of the weights) between the vertexes of the two corresponding clusters.

Author(s)

Élise Maigné <elise.maigne@inrae.fr>
Madalina Olteanu <olteanu@ceremade.dauphine.fr>
Nathalie Vialaneix <nathalie.vialaneix@inrae.fr>

References

Anderson M.J. (2001). A new method for non-parametric multivariate analysis of variance. *Austral Ecology*, **26**, 32-46.

Olteanu M., Villa-Vialaneix N. (2015) Using SOMbrero for clustering and visualizing graphs. *Journal de la Societe Francaise de Statistique*, **156**, 95-119.

See Also

hclust, cutree, trainSOM, plot.somRes

Examples

```
set.seed(11051729)
my.som <- trainSOM(x.data = iris[,1:4])
# choose the number of super-clusters
sc <- superClass(my.som)
plot(sc)
# cut the clustering
sc <- superClass(my.som, k = 4)
summary(sc)
plot(sc)
plot(sc, type = "grid")
plot(sc, what = "obs", type = "hitmap")
```

---

trainSOM *Run the SOM algorithm*

Description

The `trainSOM` function returns a `somRes` class object which contains the outputs of the algorithm.

Usage

```
trainSOM(x.data, ...)

## S3 method for class 'somRes'
print(x, ...)

## S3 method for class 'somRes'
summary(object, ...)
```

Arguments

`x.data`	a data frame or matrix containing the observations to be mapped on the grid by the SOM algorithm.
`...`	Further arguments to be passed to the function `initSOM` for specifying the parameters of the algorithm. The default values of the arguments `maxit` and `dimension` are calculated according to the SOM type if the user does not set them: • `maxit` is equal to (number of rows+number of columns)*5 if the SOM type is `korresp`. It is equal to number of rows*5 in all other SOM types • `dimension`: for a `korresp` SOM, is approximately equal to the square root of the number of observations to be classified divided by 10 but it is never smaller than 5 or larger than 10.
`x`	an object of class `somRes`.
`object`	an object of class `somRes`.

Details

The version of the SOM algorithm implemented in this package is the stochastic version.

Several variants able to handle non-vectorial data are also implemented in their stochastic versions: `type="korresp"` for contingency tables, as described in Cottrell et al. (2004) (with weights as in Cottrell and Letrémy, 2005a); `type = "relational"` for dissimilarity matrices, as described in Olteanu et al. (2015), with the fast implementation introduced in Mariette *et al.* (2017).

Missing values are handled as described in Cottrell et al. (2005b), not using missing entries of the selected observation during winner computation or prototype updates. This allows to proceed with the imputation of missing entries with the corresponding entries of the cluster prototype (with `impute`).

`summary` produces a complete summary of the results that displays the parameters of the SOM, quality criteria and ANOVA. For `type = "numeric"` the ANOVA is performed for each input variable and test the difference of this variable across the clusters of the map. For `type = "relational"` a dissimilarity ANOVA is performed (Anderson, 2001), except that in the present version, a crude estimate of the p-value is used which is based on the Fisher distribution and not on a permutation test.

Value

The `trainSOM` function returns an object of class `somRes` which contains the following components:

- clustering the final classification of the data.
- prototypes the final coordinates of the prototypes.
- energy the final energy of the map. For the numeric case, energy with data having missing entries is based on data imputation as described in Cottrell and Letrémy (2005b).
- backup a list containing some intermediate backups of the prototypes coordinates, clustering, energy and the indexes of the recorded backups, if `nb.save` is set to a value larger than 1.
- data the original dataset used to train the algorithm.
- parameters a list of the map's parameters, which is an object of class `paramSOM` as produced by the function `initSOM`.

The function `summary.somRes` also provides an ANOVA (ANalysis Of VAriance) of each input numeric variables in function of the map's clusters. This is helpful to see which variables participate to the clustering.

Note

Warning! Recording intermediate backups with the argument `nb.save` can strongly increase the computational time since calculating the entire clustering and the energy is time consuming. Use this option with care and only when it is strictly necessary.

Author(s)

Élise Maigné <elise.maigne@inrae.fr>
Jérome Mariette <jerome.mariette@inrae.fr>
Madalina Olteanu <olteanu@ceremade.dauphine.fr>
Fabrice Rossi <fabrice.rossi@apiacoa.org>
Nathalie Vialaneix <nathalie.vialaneix@inrae.fr>

References

Anderson M.J. (2001). A new method for non-parametric multivariate analysis of variance. *Austral Ecology*, **26**, 32-46.

Kohonen T. (2001) *Self-Organizing Maps*. Berlin/Heidelberg: Springer-Verlag, 3rd edition.

Cottrell M., Ibbou S., Letrémy P. (2004) SOM-based algorithms for qualitative variables. *Neural Networks*, **17**, 1149-1167.

Cottrell M., Letrémy P. (2005a) How to use the Kohonen algorithm to simultaneously analyse individuals in a survey. *Neurocomputing*, **21**, 119-138.

Cottrell M., Letrémy P. (2005b) Missing values: processing with the Kohonen algorithm. *Proceedings of Applied Stochastic Models and Data Analysis (ASMDA 2005)*, 489-496.

Olteanu M., Villa-Vialaneix N. (2015) On-line relational and multiple relational SOM. *Neurocomputing*, **147**, 15-30.

Mariette J., Rossi F., Olteanu M., Mariette J. (2017) Accelerating stochastic kernel SOM. In: M. Verleysen, *XXVth European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN 2017)*, i6doc, Bruges, Belgium, 269-274.

See Also

See `initSOM` for a description of the parameters to pass to the trainSOM function to change its behavior and `plot.somRes` to plot the outputs of the algorithm.

Examples

```
# Run trainSOM algorithm on the iris data with 500 iterations
iris.som <- trainSOM(x.data=iris[,1:4])
iris.som
summary(iris.som)
```

Index