Package ‘missSOM’

October 13, 2022

Version 1.0.1

Title Self-Organizing Maps with Built-in Missing Data Imputation

Description The Self-Organizing Maps with Built-in Missing Data Imputation. Missing values are imputed and regularly updated during the online Kohonen algorithm. Our method can be used for data visualisation, clustering or imputation of missing data. It is an extension of the online algorithm of the 'kohonen' package. The method is described in the article ``Self-Organizing Maps for Exploration of Partially Observed Data and Imputation of Missing Values'' by S. Rejeb, C. Duveau, T. Rebafka (2022) <arXiv:2202.07963>.

License GPL (>= 2)

Depends R (>= 4.0.0)

Imports Rcpp (>= 1.0.7), kpodclustr

LinkingTo Rcpp

NeedsCompilation yes

RoxygenNote 7.1.2

Encoding UTF-8

Author Sara Rejeb [aut, cre],
Tabea Rebafka [ctb],
Catherine Duveau [ctb],
Ron Wehrens [cph] (Author of included functions from the 'kohonen' package),
Johannes Kruisselbrink [cph] (Author of included functions from the 'kohonen' package)

Maintainer Sara Rejeb <sara.rejeb@live.fr>

Repository CRAN

Date/Publication 2022-05-05 06:20:06 UTC

R topics documented:

`imputeSOM` *The Self-Organizing Maps with Built-in Missing Data Imputation.*

Description

`imputeSOM` is an extension of the online algorithm of the 'kohonen' package where missing data are imputed during the algorithm. All missing values are first imputed with initial values such as the mean of the observed variables.

Usage

```
imputeSOM(
  data,
  grid = somgrid(),
  rlen = 100,
  alpha = c(0.05, 0.01),
  radius = quantile(nhbrdist, 2/3),
  maxNA.fraction = 1,
  keep.data = TRUE,
  dist.fcts = NULL,
  init
)
```

Arguments

`data`	a `matrix` or `data.frame` with continuous variables containing the observations to be mapped on the grid by the kohonen algorithm, even if there are incomplete.
`grid`	a grid for the codebook vectors: see `somgrid`.
`rlen`	the number of times the complete data set will be presented to the network.
`alpha`	learning rate, a vector of two numbers indicating the amount of change. Default is to decline linearly from 0.05 to 0.01 over `rlen` updates.
`radius`	the radius of the neighbourhood, either given as a single number or a vector (start, stop). If it is given as a single number the radius will change linearly from `radius` to zero; as soon as the neighbourhood gets smaller than one only the winning unit will be updated. Note that the default before version 3.0 was to

	run from `radius` to `-radius`. If nothing is supplied, the default is to start with a value that covers 2/3 of all unit-to-unit distances.
`maxNA.fraction`	the maximal fraction of values that may be NA to prevent the column to be removed.
`keep.data`	if TRUE, return original data and mapping information. If FALSE, only return the trained map (in essence the codebook vectors).
`dist.fcts`	distance function to be used for the data. Admissable values currently are "sumofsquares", "euclidean" and "manhattan. Default is to use "sumofsquares".
`init`	a `matrix` or `data.frame` corresponding to the initial values for the codebook vectors. It should have the same number of variables (columns) as the data. The number of rows corresponding to the number of units in the map.

Value

An object of class "missSOM" with components

`data`	Data matrix, only returned if `keep.data == TRUE`.
`ximp`	Imputed data matrix.
`unit.classif`	Winning units for data objects, only returned if `keep.data == TRUE`.
`distances`	Distances of objects to their corresponding winning unit, only returned if `keep.data == TRUE`.
`grid`	The grid, an object of class `somgrid`.
`codes`	A list of matrices containing codebook vectors.
`alpha, radius`	Input arguments presented to the function.
`maxNA.fraction`	The maximal fraction of values that may be NA to prevent the column to be removed.
`dist.fcts`	The distance function used for the data.

See Also

somgrid, `plot.missSOM`, `map.missSOM`

Examples

```
data(wines)

## Data with no missing values
som.wines <- imputeSOM(scale(wines), grid = somgrid(5, 5, "hexagonal"))
summary(som.wines)
print(dim(som.wines$data))

## Data with missing values
X <- scale(wines)
missing_obs <- sample(1:nrow(wines), 10, replace = FALSE)
X[missing_obs, 1:2] <- NaN
som.wines <- imputeSOM(X, grid = somgrid(5, 5, "hexagonal"))
summary(som.wines)
```

```
print(dim(som.wines$ximp))
print(sum(is.na(som.wines$ximp)))
```

`map` *Map data to a supervised or unsupervised SOM*

Description

Map a data onto a trained SOM.

Usage

```
map(x, ...)

## S3 method for class 'missSOM'
map(x, newdata, maxNA.fraction = x$maxNA.fraction, ...)
```

Arguments

`x`	an object of class `missSOM`.
`...`	Currently ignored.
`newdata`	a `matrix` or `data.frame`, equal to the `data` argument of the `imputeSOM` function.
`maxNA.fraction`	parameters that usually will be taken from the `x` object, but can be supplied by the user as well. Note that it is not possible to change distance functions from the ones used in training the map. See `imputeSOM` for more information.

Value

A list with elements

`unit.classif`	a vector of units that are closest to the objects in the data.
`dists`	distances of the objects to the closest units. Distance measures are the same ones used in training the map.

See Also

`imputeSOM`

Examples

```
data(wines)
set.seed(7)

training <- sample(nrow(wines), 150)
Xtraining <- scale(wines[training, ])
somnet <- imputeSOM(Xtraining, somgrid(5, 5, "hexagonal"))

map(somnet, scale(wines[-training, ],
                  center=attr(Xtraining, "scaled:center"),
                  scale=attr(Xtraining, "scaled:scale")))
```

`missSOM` *missSOM*

Description

The Self-Organizing Maps with Built-in Missing Data Imputation. Missing values are imputed and regularly updated during the online Kohonen algorithm. Our method can be used for data visualisation, clustering or imputation of missing data. It is an extension of the online algorithm of the kohonen package.

Details

Self-Organizing Maps with Built-in Missing Data Imputation

Author(s)

you <youremail>

`nir` *Title Near-infrared data with temperature effects*

Description

A data object containing near-infrared spectra of ternary mixtures of ethanol, water and iso-propanol, measured at five different temperatures (30, 40, ..., 70 degrees Centigrade).

Author(s)

My Name <`blahblah@roxygen.org`>

References

F. Wulfert , W.Th. Kok, A.K. Smilde: Anal. Chem. 1998, 1761-1767

object.distances *Calculate distances between object vectors in a SOM*

Description

This function calculates the distance between objects using the distance functions, weights and other attributes of a trained SOM. This function is used in the calculation of the U matrix in function `plot.missSOM` using the `type = "dist.neighbours"` argument.

Usage

```
object.distances(kohobj, type = c("data", "ximp", "codes"))
```

Arguments

`kohobj`	An object of class `missSOM`.
`type`	Whether to calculate distances between the data objects, or the codebook vectors.

Value

An object of class `dist`, which can be directly fed into (e.g.) a hierarchical clustering.

See Also

`unit.distances`, `imputeSOM`

Examples

```
data(wines)

## Data with no missing values
set.seed(7)
sommap <- imputeSOM(scale(wines), grid = somgrid(6, 4, "hexagonal"))
obj.dists <- object.distances(sommap, type = "data")
code.dists <- object.distances(sommap, type = "codes")

## Data with missing values
X <- scale(wines)
X[1:5, 1] <- NaN
sommap <- imputeSOM(X, grid = somgrid(6, 4, "hexagonal"))
obj.dists <- object.distances(sommap, type = "ximp")
code.dists <- object.distances(sommap, type = "codes")
```

`plot.missSOM`	*Plot missSOM object*

Description

Plot objects of class missSOM. Several types of plots are supported.

Usage

```
## S3 method for class 'missSOM'
plot(
  x,
  type = c("codes", "changes", "counts", "dist.neighbours", "mapping", "property",
    "quality"),
  classif = NULL,
  labels = NULL,
  pchs = NULL,
  main = NULL,
  palette.name = NULL,
  ncolors,
  bgcol = NULL,
  zlim = NULL,
  heatkey = TRUE,
  property,
  codeRendering = NULL,
  keepMargins = FALSE,
  heatkeywidth = 0.2,
  shape = c("round", "straight"),
  border = "black",
  na.color = "gray",
  ...
)

add.cluster.boundaries(x, clustering, lwd = 5, ...)

## S3 method for class 'missSOM'
identify(x, ...)
```

Arguments

`x`	missSOM object.
`type`	type of plot. (Wow!)
`classif`	classification object or vector of unit numbers. Only needed if `type` equals "mapping" and "counts".
`labels`	labels to plot when `type` equals "mapping".
`pchs`	symbols to plot when `type` equals "mapping".

`main`	title of the plot.
`palette.name`	colors to use as unit background for "codes", "counts", "prediction", "property", and "quality" plotting types.
`ncolors`	number of colors to use for the unit backgrounds. Default is 20 for continuous data.
`bgcol`	optional argument to colour the unit backgrounds for the "mapping" and "codes" plotting type. Defaults to "gray" and "transparent" in both types, respectively.
`zlim`	optional range for color coding of unit backgrounds.
`heatkey`	whether or not to generate a heatkey at the left side of the plot in the "property" and "counts" plotting types.
`property`	values to use with the "property" plotting type.
`codeRendering`	How to show the codes. Possible choices: "segments", "stars" and "lines".
`keepMargins`	if `FALSE` (the default), restore the original graphical parameters after plotting the kohonen map. If `TRUE`, one retains the map coordinate system so that one can add symbols to the plot, or map unit numbers using the `identify` function.
`heatkeywidth`	width of the colour key; the default of 0.2 should work in most cases but in some cases, e.g. when plotting multiple figures, it may need to be adjusted.
`shape`	kind shape to be drawn: "round" (circle) or "straight". Choosing "straight" produces a map of squares when the grid is "rectangular", and produces a map of hexagons when the grid is "hexagonal".
`border`	color of the shape's border.
`na.color`	background color matching NA - default "gray".
`...`	other graphical parameters.
`clustering`	cluster labels of the map units.
`lwd`	other graphical parameters.

Details

Several different types of plots are supported:

"changes" shows the mean distance to the closest codebook vector during training.

"codes" shows the codebook vectors.

"counts" shows the number of objects mapped to the individual units. Empty units are depicted in gray.

"dist.neighbours" shows the sum of the distances to all immediate neighbours. This kind of visualisation is also known as a U-matrix plot. Units near a class boundary can be expected to have higher average distances to their neighbours.

"mapping" shows where objects are mapped. It needs the "classif" argument, and a "labels" or "pchs" argument.

"property" properties of each unit can be calculated and shown in colour code. It can be used to visualise the similarity of one particular object to all units in the map, to show the mean similarity of all units and the objects mapped to them, etcetera. The parameter `property` contains the numerical values. See examples below.

"quality" shows the mean distance of objects mapped to a unit to the codebook vector of that unit. The smaller the distances, the better the objects are represented by the codebook vectors.

Function `identify.missSOM` shows the number of a unit that is clicked on with the mouse. The tolerance is calculated from the ratio of the plotting region and the user coordinates, so clicking at any place within a unit should work.

Function `add.cluster.boundaries` will add to an existing plot of a map thick lines, visualizing which units would be clustered together. In toroidal maps, boundaries at the edges will only be shown on the top and right sides to avoid double boundaries.

Value

Several types of plots return useful values (invisibly): the `"counts"`, `"dist.neighbours"`, and `"quality"` return vectors corresponding to the information visualized in the plot (unit background colours and heatkey).

See Also

`imputeSOM`

Examples

```
data(wines)
set.seed(7)
SOM.map <- imputeSOM(scale(wines), grid = somgrid(5, 5, "hexagonal"), rlen=100)
plot(SOM.map, type="changes")
counts <- plot(SOM.map, type="counts", shape = "straight")
## show both sets of codebook vectors in the map
plot(SOM.map, type="codes", main = c("Codes X"))

oldpar <- par(mfrow = c(1,2))
similarities <- plot(SOM.map, type="quality", palette.name = terrain.colors)
plot(SOM.map, type="mapping",
     labels = as.integer(vintages), col = as.integer(vintages),
     main = "mapping plot")
par(oldpar)

## Show 'component planes'
set.seed(7)
sommap <- imputeSOM(scale(wines), grid = somgrid(6, 4, "hexagonal"))
plot(sommap, type = "property", property = sommap$codes[,1],
     main = colnames(sommap$codes)[1])

## Show the U matrix
Umat <- plot(sommap, type="dist.neighbours", main = "SOM neighbour distances")
## use hierarchical clustering to cluster the codebook vectors
som.hc <- cutree(hclust(object.distances(sommap, "codes")), 5)
add.cluster.boundaries(sommap, som.hc)

## and the same for rectangular maps
set.seed(7)
sommap <- imputeSOM(scale(wines),grid = somgrid(6, 4, "rectangular"))
```

```
plot(sommap, type="dist.neighbours", main = "SOM neighbour distances")
## use hierarchical clustering to cluster the codebook vectors
som.hc <- cutree(hclust(object.distances(sommap, "codes")), 5)
add.cluster.boundaries(sommap, som.hc)
```

somgrid *SOM-grid related functions*

Description

Function somgrid (modified from the version in the class package) sets up a grid of units, of a specified size and topology. Distances between grid units are calculated by function unit.distances.

Usage

```
somgrid(
  xdim = 8,
  ydim = 6,
  topo = c("rectangular", "hexagonal"),
  neighbourhood.fct = c("bubble", "gaussian"),
  toroidal = FALSE
)

unit.distances(grid, toroidal)
```

Arguments

xdim	dimensions of the grid.
ydim	dimensions of the grid.
topo	choose between a hexagonal or rectangular topology.
neighbourhood.fct	choose between bubble and gaussian neighbourhoods when training a SOM.
toroidal	logical, whether the grid is toroidal or not. If not provided to the unit.distances function, the information in the grid object will be used.
grid	an object of class somgrid.

Value

Function somgrid returns an object of class "somgrid", with elements pts, and the input arguments to the function.

Function unit.distances returns a (symmetrical) matrix containing distances. When grid$n.hood equals "circular", Euclidean distances are used; for grid$n.hood is "square" maximum distances. For toroidal maps (joined at the edges) distances are calculated for the shortest path.

Examples

```
mygrid <- somgrid(5, 5, "hexagonal")
fakesom <- list(grid = mygrid)
class(fakesom) <- "missSOM"

oldpar <- par(mfrow = c(2,1))
dists <- unit.distances(mygrid)
plot(fakesom, type="property", property = dists[1,],
     main="Distances to unit 1", zlim=c(0,6),
     palette = rainbow, ncolors = 7)
dists <- unit.distances(mygrid, toroidal=TRUE)
plot(fakesom, type="property", property = dists[1,],
     main="Distances to unit 1 (toroidal)", zlim=c(0,6),
     palette = rainbow, ncolors = 7)
par(oldpar)
```

`summary.missSOM` *Summary and print methods for missSOM objects*

Description

Summary and print methods for `missSOM` objects. The `print` method shows the dimensions and the topology of the map; if information on the training data is included, the `summary` method additionally prints information on the size of the data, the distance functions used, and the mean distance of an object to its closest codebookvector, which is an indication of the quality of the mapping.

Usage

```
## S3 method for class 'missSOM'
summary(object, ...)
## S3 method for class 'missSOM'
print(x, ...)

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

Arguments

`object`	a `missSOM` object
`...`	Not used.
`x`	a kohonen object

Value

No return a value.

See Also

`imputeSOM`

Examples

```
data(wines)
som.wines <- imputeSOM(scale(wines), grid = somgrid(5, 5, "hexagonal"))
som.wines
summary(som.wines)
```

`tricolor` *Provides smooth unit colors for SOMs*

Description

Function provides colour values for SOM units in such a way that the colour changes smoothly in every direction.

Usage

```
tricolor(grid, phis = c(0, 2 * pi/3, 4 * pi/3), offset = 0)
```

Arguments

`grid`	An object of class `somgrid`, such as the `grid` element in a `kohonen` object.
`phis`	A vector of three rotation angles. Values for red, green and blue are given by the y-coordinate of the units after rotation with these three angles, respectively. The default corresponds to (approximate) red colour of the middle unit in the top row, and pure green and blue colours in the bottom left and right units, respectively. In case of a triangular map, the top unit is pure red.
`offset`	Defines the minimal value in the RGB colour definition (default is 0). By supplying a value in the range [0, .9], pastel-like colours are provided.

Value

Returns a matrix with three columns corresponding to red, green and blue. This can be used in the `rgb` function to provide colours for the units.

See Also

`plot.missSOM`

Examples

```
data(wines)
som.wines <- imputeSOM(wines, grid = somgrid(5, 5, "hexagonal"))

colour1 <- tricolor(som.wines$grid)
plot(som.wines, "mapping", bg = rgb(colour1))
colour2 <- tricolor(som.wines$grid, phi = c(pi/6, 0, -pi/6))
plot(som.wines, "mapping", bg = rgb(colour2))
colour3 <- tricolor(som.wines$grid, phi = c(pi/6, 0, -pi/6), offset = .5)
plot(som.wines, "mapping", bg = rgb(colour3))
```

`wines` *Wine data*

Description

A data frame containing 177 rows and thirteen columns; object `vintages` contains the class labels.

These data are the results of chemical analyses of wines grown in the same region in Italy (Piedmont) but derived from three different cultivars: Nebbiolo, Barberas and Grignolino grapes. The wine from the Nebbiolo grape is called Barolo. The data contain the quantities of several constituents found in each of the three types of wines, as well as some spectroscopic variables.

Author(s)

My Name `<blahblah@roxygen.org>`

Source

`http://kdd.ics.uci.edu`

References

M. Forina, C. Armanino, M. Castino and M. Ubigli. Vitis, 25:189-201 (1986)

`yeast` *Title Yeast cell-cycle data*

Description

Microarray cell-cycle data for 800 yeast genes, arrested with six different methods, arranged in a list. Additional class information is present as well.

Author(s)

My Name `<blahblah@roxygen.org>`

References

P. Spellman et al., Mol. Biol. Cell 9, 3273-3297 (1998)

Index