Package 'kohonen'

January 27, 2015

Version 2.0.15

Title Supervised and unsupervised self-organising maps

Author Ron Wehrens

Maintainer Ron Wehrens <ron.wehrens@gmail.com>

Description Supervised and unsupervised self-organising maps

License GPL (>= 2)

Depends R (>= 2.6.0), class, MASS

NeedsCompilation yes

Repository CRAN

Date/Publication 2014-08-04 12:15:23

R topics documented:

bdk *Supervised version of Kohonen's self-organising maps*

Description

Supervised version of self-organising maps for mapping high-dimensional spectra or patterns to 2D: the Bi-Directional Kohonen map. This is an alternating training of the X-space and the Y-space of the map, where for updating the X-space more weight is given to the features in Y, and vice versa. Weights start by default with values of (0.75, 0.25) and during training go to (0.5, 0.5). Prediction is done only using the X-space. For continuous Y, the Euclidean distance is used; for categorical Y the Tanimoto distance.

Usage

```
bdk(data, Y, grid=somgrid(), rlen = 100, alpha = c(0.05, 0.01),
    radius = quantile(nhbrdist, 0.67) * c(1, -1), xweight = 0.75,
    contin, toroidal = FALSE, n.hood, keep.data = TRUE)
```

Arguments

`data`	a matrix, with each row representing an object.
`Y`	property that is to be modelled. In case of classification, Y is a matrix with exactly one '1' in each row indicating the class, and zeros elsewhere. For prediction of continuous properties, Y is a vector. A combination is possible, too, but one then should take care of appropriate scaling.
`grid`	a grid for the representatives: 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 run from the given number to the negative value of that number; as soon as the neighbourhood gets smaller than one only the winning unit will be updated. The default is to start with a value that covers 2/3 of all unit-to-unit distances.
`xweight`	the initial weight given to the X map in the calculation of distances for updating Y, and to the Y map for updating X. This will linearly go to 0.5 during training. Defaults to 0.75.
`contin`	parameter indicating whether Y is continuous or categorical. The default is to check whether all row sums of Y equal 1: in that case `contin` is `FALSE`.
`toroidal`	if `TRUE`, the edges of the map are joined. Note that in a hexagonal toroidal map, the number of rows must be even.
`n.hood`	the shape of the neighbourhood, either "circular" or "square". The latter is the default for rectangular maps, the former for hexagonal maps.
`keep.data`	save data in return value.

Value

an object of class "kohonen" with components

`data`	data matrix, only returned if `keep.data == TRUE`.
`Y`	Y, only returned if `keep.data == TRUE`.
`contin`	parameter indicating whether Y is continuous or categorical.
`grid`	the grid, an object of class "somgrid".
`codes`	list of two matrices, containing codebook vectors for X and Y, respectively.
`changes`	matrix containing two columns of mean average deviations from code vectors. Column 1 contains deviations used for updating Y; column 2 for updating X.
`toroidal`	whether a toroidal map is used.
`unit.classif`	winning units for all data objects, only returned if `keep.data == TRUE`.
`distances`	distances of objects to their corresponding winning unit, only returned if `keep.data == TRUE`.
`method`	the type of som, here "bdk"

Author(s)

Ron Wehrens

References

W.J. Melssen, R. Wehrens, and L.M.C. Buydens. Chemom. Intell. Lab. Syst., 83, 99-113 (2006).

See Also

`som`, `xyf`, `plot.kohonen`, `predict.kohonen`

Examples

```
### Wine example
data(wines)
set.seed(7)

training <- sample(nrow(wines), 120)
Xtraining <- scale(wines[training,])
Xtest <- scale(wines[-training,],
               center = attr(Xtraining, "scaled:center"),
               scale = attr(Xtraining, "scaled:scale"))

bdk.wines <- bdk(Xtraining,
                 factor(wine.classes[training]),
                 grid = somgrid(5, 5, "hexagonal"))

bdk.prediction <- predict(bdk.wines, newdata=Xtest)
table(wine.classes[-training], bdk.prediction$prediction)
```

`check.whatmap` *Check the validity of a whatmap argument*

Description

Not meant to be called directly by the user.

Usage

```
check.whatmap(x, whatmap)
```

Arguments

`x`	Either a `kohonen` object from `supersom`, or a list of data matrices that can be used as input data for `supersom`.
`whatmap`	An indication of a subset of the data; either by naming the elements, or giving indices. If `whatmap` equals NULL, no selection is performed.

Value

Returns a numerical vector with the indices of the selected layers.

Author(s)

Ron Wehrens

`classvec2classmat` *Convert a classification vector into a matrix or the other way around.*

Description

Functions toggle between a matrix representation, where class membership is indicated with one '1' and for the rest zeros at each row, and an class vector (maybe integers or class names). The classification matrix contains one column per class. Conversion from a class matrix to a class vector assigns each row to the column with the highest value. An optional argument can be used to assign only those objects that have a probability higher than a certain threshold (default is 0).

Usage

```
classvec2classmat(yvec)
classmat2classvec(ymat, threshold=0)
```

Arguments

`yvec`	class vector. Usually integer values, but other types are also allowed.
`ymat`	class matrix: every column corresponds to a class.
`threshold`	only classify into a class if the probability is larger than this threshold.

Value

`classvec2classmat` returns the classification matrix, where each column consists of zeros and ones; `classmat2classvec` returns a class vector (integers).

Author(s)

Ron Wehrens

See Also

`bdk`, `xyf`

Examples

```
classes <- c(rep(1, 5), rep(2, 7), rep(3, 9))
classmat <- classvec2classmat(classes)
classmat
classmat2classvec(classmat)
```

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

Description

Map a data matrix onto a trained SOM.

Usage

```
## S3 method for class 'kohonen'
map(x, newdata, whatmap = NULL, weights,
            scale.distances = (nmaps > 1), ...)
```

Arguments

`x`	A trained supervised or unsupervised SOM obtained from functions `som`, `xyf`, or `bdk`.
`newdata`	Data matrix, with rows corresponding to objects.
`whatmap`	For `supersom` maps: the layers to take into account.
`weights`	For `supersom` maps: weights of the layers that are used for mapping.

`scale.distances`	whether to rescale distances per layer in the case of `supersom` maps (default): if `TRUE` the maximal distance of each layer equals one. If the absolute values of the distances per layer should be used, this argument should be set to `FALSE`. Note that in that case, when mapping the training data, the result returned by `map.kohonen` will differ from the mapping present in the map.
`...`	Currently ignored.

Value

A list with elements

`unit.classif`	a vector of units that are closest to the objects in the data matrix.
`dists`	distances (currently only Euclidean distances) of the objects to the units.
`whatmap,weights,scale.distances`	Values used for these arguments.

Author(s)

Ron Wehrens

See Also

`predict.kohonen`

Examples

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

training <- sample(nrow(wines), 120)
Xtraining <- scale(wines[training, ])
somnet <- som(Xtraining, somgrid(5, 5, "hexagonal"))

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

`nir` *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).

References

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

Examples

```
data(nir)

set.seed(3)
nirnet <- xyf(data = nir$spectra[nir$training,],
              Y = nir$composition[nir$training,],
              xweight=.75,
              grid = somgrid(6, 6, "hexagonal"), rlen=500)
plot(nirnet, "counts", main="Counts")

## Focus on compound 2 (water):
dev.new(width = 14)
par(mfrow = c(1,2))
set.seed(13)
nirnet <- xyf(data = nir$spectra[nir$training,],
              Y = nir$composition[nir$training, 2],
              grid = somgrid(6, 6, "hexagonal"), rlen=500)
water.xyf <- predict(nirnet)$prediction
plot(nirnet, "property", property = water.xyf,
     main="Prediction of water content")
## Plot temperatures as circles
symbols(nirnet$grid$pts[nirnet$unit.classif,] +
        matrix(rnorm(sum(nir$training)*2, sd=.1), ncol=2),
circles = (nir$temperature[nir$training] - 20)/250,
inches = FALSE, add = TRUE)

## Model temperatures
set.seed(13)
nirnet2 <- xyf(data = nir$spectra[nir$training,],
               Y = nir$temperature[nir$training],
               xweight=.25,
               grid = somgrid(6, 6, "hexagonal"), rlen=500)
temp.xyf <- predict(nirnet2)$prediction

plot(nirnet2, "property", property = temp.xyf,
     palette.name = rainbow,
     main="Prediction of temperatures")
## Plot concentrations of water as circles
symbols(nirnet2$grid$pts[nirnet2$unit.classif,] +
        matrix(rnorm(sum(nir$training)*2, sd=.1), ncol=2),
circles = 0.05 + 0.4 * nir$composition[nir$training,2],
inches = FALSE, add = TRUE)
```

---

plot.kohonen *Plot kohonen object*

Description

Plot self-organising map, obtained from function kohonen. Several types of plots are supported.

Usage

```
## S3 method for class 'kohonen'
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, contin,
     whatmap = NULL, codeRendering = NULL, keepMargins = FALSE,
     heatkeywidth = .2, ...)
## S3 method for class 'kohonen'
identify(x, ...)
add.cluster.boundaries(x, clustering, lwd = 5, ...)
```

Arguments

`x`	kohonen object.
`type`	type of plot. (Wow!)
`classif`	classification object, as returned by `predict.kohonen`, 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, and the number of distinct values (if less than 20) for categorical 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.
`contin`	whether or not the data should be seen as discrete (i.e. classes) or continuous in nature. Only relevant for the colour keys of plots of supervised networks. Note that this is different from the `contin` argument in the `xyf`, `bdk` and `supersom` functions.
`whatmap`	For `supersom` maps and a "codes" plot: what maps to show.
`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.
`lwd, ...`	other graphical parameters.
`clustering`	cluster labels of the map units.

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. Only available for the "som" and "supersom" maps, for the moment.

"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.kohonen` 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.

Author(s)

Ron Wehrens

See Also

`som`, `bdk`, `xyf`

Examples

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

kohmap <- xyf(scale(wines), classvec2classmat(wine.classes),
              grid = somgrid(5, 5, "hexagonal"), rlen=100)
```

```
plot(kohmap, type="changes")
plot(kohmap, type="codes", main = c("Codes X", "Codes Y"))
plot(kohmap, type="counts")

## palette suggested by Leo Lopes
coolBlueHotRed <- function(n, alpha = 1) {
  rainbow(n, end=4/6, alpha=alpha)[n:1]
}
plot(kohmap, type="quality", palette.name = coolBlueHotRed)
plot(kohmap, type="mapping",
     labels = wine.classes, col = wine.classes+1,
     main = "mapping plot")

## add background colors to units according to their predicted class labels
xyfpredictions <- classmat2classvec(predict(kohmap)$unit.predictions)
bgcols <- c("gray", "pink", "lightgreen")
plot(kohmap, type="mapping", col = wine.classes+1,
     pchs = wine.classes, bgcol = bgcols[as.integer(xyfpredictions)],
     main = "another mapping plot")

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

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

## and the same for rectangular maps
set.seed(7)
sommap <- som(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(dist(sommap$codes)), 5)
add.cluster.boundaries(sommap, som.hc)
```

`predict.kohonen` *Predict properties using a trained Kohonen map*

Description

Map objects to a trained Kohonen map, and return for each object the property associated with the corresponding winning unit. For `som` and `supersom` maps, the unit properties are calculated using explicit arguments `trainX` and `trainY`; for `xyf` and `bdk` maps, the predicted properties are the Y-codebookvectors. Note that in the latter case only the X-space is used for prediction.

Usage

```
## S3 method for class 'kohonen'
predict(object, newdata, trainX, trainY, unit.predictions,
                  threshold = 0, whatmap = NULL, weights = 1, ...)
```

Arguments

`object`	Trained network.
`newdata`	Data matrix for which predictions are to be made. If not given, defaults to the training data (when available).
`trainX`	Training data for obtaining predictions for unsupervised maps; necessary for `som` maps trained with the `keep.data = FALSE` option.
`trainY`	Values for the dependent variable for the training data; necessary for `som` and `supersom` maps.
`unit.predictions`	Possible override of the predictions for each unit.
`threshold`	Used in class predictions; see `classmat2classvec`.
`whatmap`	For `supersom` maps: what layers to use in the mapping.
`weights`	For `supersom` maps: weights of layers uses in the mapping.
`...`	Currently not used.

Value

Returns a list with components

`prediction`	predicted values for the properties of interest. When multiple values are predicted, this element is a list, otherwise a vector or a matrix.
`unit.classif`	unit numbers to which objects in the data matrix are mapped.
`unit.predictions`	mean values associated with map units. Again, when multiple properties are predicted, this is a list.

Author(s)

Ron Wehrens

See Also

`som`,`xyf`,`bdk`, `supersom`, `map`

Examples

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

training <- sample(nrow(wines), 120)
Xtraining <- scale(wines[training, ])
```

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

som.wines <- som(Xtraining, grid = somgrid(5, 5, "hexagonal"))

som.prediction <- predict(som.wines, newdata = Xtest,
          trainX = Xtraining,
          trainY = factor(wine.classes[training]))
table(wine.classes[-training], som.prediction$prediction)

### more complicated examples
data(yeast)

### only consider complete cases
missings <- (apply(cbind(yeast$alpha, yeast$cdc15), 1,
                   function(x) any(is.na(x))))

yeast2 <- list(alpha = yeast$alpha[!missings,],
               cdc15 = yeast$cdc15[!missings,],
               class = yeast$class[!missings])

set.seed(7)
training.indices <- sample(nrow(yeast2$alpha), 300)
training <- rep(FALSE, nrow(yeast2$alpha))
training[training.indices] <- TRUE

## unsupervised mapping
yeast2.som <- som(yeast2$alpha[training,], somgrid(4, 6, "hexagonal"))
yeast2.som.prediction <- predict(yeast2.som,
                                 newdata = yeast2$alpha[!training,],
                                 trainY = yeast2$class[training])
table(yeast2$class[!training], yeast2.som.prediction$prediction)

## supervised mapping (XYF) - trainY is no longer necessary
yeast2.xyf <- xyf(yeast2$alpha[training,], yeast2$class[training],
                  somgrid(4, 6, "hexagonal"))
yeast2.xyf.prediction <- predict(yeast2.xyf,
                                 newdata = yeast2$alpha[!training,])
table(yeast2$class[!training], yeast2.xyf.prediction$prediction)

## supervised mapping (BDK)
yeast2.bdk <- bdk(yeast2$alpha[training,], yeast2$class[training],
                  somgrid(4, 6, "hexagonal"))
yeast2.bdk.prediction <- predict(yeast2.bdk,
                                 newdata = yeast2$alpha[!training,])
table(yeast2$class[!training], yeast2.bdk.prediction$prediction)

## unsupervised mapping (supersom): prediction of data layer not used
## in training
yeast2.ssom <- supersom(lapply(yeast2, function(x) subset(x, training)),
                        grid = somgrid(4, 6, "hexagonal"),
                        whatmap = 1)
```

```
yeast2.ssom.prediction <- predict(yeast2.ssom,
                                  newdata = lapply(yeast2,
                                    function(x) subset(x, !training)),
                                  trainY = list(class = yeast2$class[training]))
table(yeast2$class[!training], yeast2.ssom.prediction$prediction)

## supervised mapping (supersom): prediction of a data layer that has
## been used in training - trainY is not necessary
yeast2.ssom2 <- supersom(lapply(yeast2, function(x) subset(x, training)),
                         grid = somgrid(4, 6, "hexagonal"),
                         whatmap = c(1,3))
yeast2.ssom2.prediction <- predict(yeast2.ssom2,
                                   newdata = lapply(yeast2,
                                     function(x) subset(x, !training)),
                                   whatmap = 1)
table(yeast2$class[!training], yeast2.ssom2.prediction$prediction)
```

---

som *Kohonen's self-organising maps*

Description

Self-organising maps for mapping high-dimensional spectra or patterns to 2D; Euclidean distance is used. Modelled after the SOM function in package `class`.

Usage

```
som(data, grid=somgrid(), rlen = 100, alpha = c(0.05, 0.01),
    radius = quantile(nhbrdist, 0.67) * c(1, -1), init,
    toroidal = FALSE, n.hood, keep.data = TRUE)
```

Arguments

`data`	a matrix, with each row representing an object.
`grid`	a grid for the representatives: 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 run from the given number to the negative value of that number; as soon as the neighbourhood gets smaller than one only the winning unit will be updated. The default is to start with a value that covers 2/3 of all unit-to-unit distances.
`init`	the initial representatives, represented as a matrix. If missing, chosen (without replacement) randomly from 'data'.
`toroidal`	if TRUE, the edges of the map are joined. Note that in a hexagonal toroidal map, the number of rows must be even.

`n.hood`	the shape of the neighbourhood, either "circular" or "square". The latter is the default for rectangular maps, the former for hexagonal maps.
`keep.data`	save data in return object.

Value

an object of class "kohonen" with components

`data`	data matrix, only returned if `keep.data == TRUE`.
`grid`	the grid, an object of class "somgrid".
`codes`	a matrix of code vectors.
`changes`	vector of mean average deviations from code vectors.
`unit.classif`	winning units for all data objects, only returned if `keep.data == TRUE`.
`distances`	distances of objects to their corresponding winning unit, only returned if `keep.data == TRUE`.
`toroidal`	whether a toroidal map is used.
`method`	the type of som, here "som".

Author(s)

Ron Wehrens

References

"Self-organizing maps", 3rd Ed., T. Kohonen, New York: Springer (2001)

See Also

`xyf`, `bdk`, `plot.kohonen`

Examples

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

training <- sample(nrow(wines), 120)
Xtraining <- scale(wines[training, ])
Xtest <- scale(wines[-training, ],
               center = attr(Xtraining, "scaled:center"),
               scale = attr(Xtraining, "scaled:scale"))

som.wines <- som(Xtraining, grid = somgrid(5, 5, "hexagonal"))

som.prediction <- predict(som.wines, newdata = Xtest,
                          trainX = Xtraining,
                          trainY = factor(wine.classes[training]))
table(wine.classes[-training], som.prediction$prediction)
```

summary.kohonen — *Summary and print methods for kohonen objects*

Description

Summary and print methods for kohonen 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 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 'kohonen'
summary(object, ...)
## S3 method for class 'kohonen'
print(x, ...)
```

Arguments

x, object	a kohonen object
...	Not used.

Author(s)

Ron Wehrens

See Also

som, xyf, bdk, supersom

Examples

```
data(wines)
xyf.wines <- xyf(scale(wines), classvec2classmat(wine.classes),
                 grid = somgrid(5, 5, "hexagonal"))
xyf.wines
summary(xyf.wines)
```

supersom *Super-organising maps*

Description

An extension of `xyf` maps to multiple data layers, possibly with different numbers of variables (though equal numbers of objects). NAs are allowed (see below). A weighted distance over all layers is calculated to determine the winning units during training.

Usage

```
supersom(data, grid=somgrid(), rlen = 100, alpha = c(0.05, 0.01),
    radius = quantile(nhbrdist, 0.67) * c(1, -1),
    contin, toroidal = FALSE, n.hood, whatmap = NULL, weights = 1,
    maxNA.fraction = .5, keep.data = TRUE)
```

Arguments

`data`	list of data matrices.
`grid`	a grid for the representatives: 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 run from the given number to the negative value of that number; as soon as the neighbourhood gets smaller than one only the winning unit will be updated. The default is to start with a value that covers 2/3 of all unit-to-unit distances.
`contin`	parameter indicating whether data are continuous or categorical, i.e. a logical vector. If only `TRUE` or `FALSE` is given, this is taken to hold for all elements in `data`. The default is to check whether row sums in the data matrices are equal to 1: in that case the corresponding `contin` element is `FALSE`.
`toroidal`	if TRUE, the edges of the map are joined. Note that in a hexagonal toroidal map, the number of rows must be even.
`n.hood`	the shape of the neighbourhood, either "circular" or "square". The latter is the default for rectangular maps, the former for hexagonal maps.
`whatmap`	For `supersom` maps: what layers to use in the mapping.
`weights`	the weights given to individual layers. Default is 1/n, with n the number of layers.
`maxNA.fraction`	the maximal fraction of values that may be NA to prevent the row or column to be removed.
`keep.data`	save data in return value.

Value

an object of class "kohonen" with components

`data`	data matrix, only returned if `keep.data == TRUE`.
`contin`	parameter indicating whether elements of `data` are continuous or categorical.
`na.rows`	indices of objects (rows) that are removed because at least one of the layers has to many NAs for these objects.
`unit.classif`	winning units for all 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.
`changes`	matrix of mean average deviations from code vectors; every map corresponds with one column.
`toroidal`	whether a toroidal map is used.
`n.hood`	the shape of the neighbourhood, either "circular" or "square". The latter is the default for rectangular maps, the former for hexagonal maps.
`weights`	For `supersom` maps: weights of layers uses in the mapping.
`whatmap`	For `supersom` maps: what layers to use in the mapping.
`method`	type of map, here "supersom".

Author(s)

Ron Wehrens

References

R. Wehrens and L.M.C. Buydens, J. Stat. Softw. 21 (5), 2007

See Also

`somgrid`, `plot.kohonen`

Examples

```
data(yeast)
yeast.supersom <- supersom(yeast, somgrid(6, 6, "hexagonal"), whatmap = 3:6)
obj.classes <- as.integer(yeast$class)
colors <- c("yellow", "green", "blue", "red", "orange")
plot(yeast.supersom, type = "mapping", col = colors[obj.classes],
     pch = obj.classes, main = "yeast data", keepMargins = TRUE)
```

topo.error *Measures of topographical error*

Description

The function implements two forms of what is known as 'topographical error', basically a measure of continuity of the mapping. This is done by assessing the distances between units containing similar patterns. See 'Details'.

Usage

```
topo.error(somobj, type = c("nodedist", "bmu"), data)
```

Arguments

`somobj`	an object of class `"kohonen"`.
`type`	which kind of topographical error to return. See 'Details'.
`data`	used in case `type = "bmu"` - the objects for which the best and second best matching units are calculated. Usually this is done for training data. If not explicitly given, the function will use any data present in `somobj`; if no data are available there either, an error message is given.

Details

Two types of topographical error are implemented, depending on the value of the `type` argument:

1. `"nodedist"`: the average distance, in terms of (x, y) coordinates in the map, between all pairs of most similar codebook vectors.
2. `"bmu"`: the average distance, in terms of (x, y) coordinates in the map, between the best matching unit and the second best matching unit, for all data points.

In both cases, low values indicate a smooth mapping, with similar objects mapped to close-by units.

Value

A number representing the topological error.

References

"Topology preservation in self-organising maps", K. Kiviluoto, IEEE Int. Conf. on Neural Networks, pp. 294-299 (1996)

"Self-organizing maps", 3rd Ed., T. Kohonen, New York: Springer (2001)

Examples

```
data(yeast)
## take only complete cases
X <- yeast[[3]][apply(yeast[[3]], 1, function(x) sum(is.na(x))) == 0,]
yeast.som <- som(X, somgrid(5, 8, "hexagonal"))

## quantization error:
mean(yeast.som$distances)
## topographical error measures:
topo.error(yeast.som, "nodedist")
topo.error(yeast.som, "bmu")
```

`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.

Author(s)

Ron Wehrens

See Also

`plot.kohonen`

Examples

```
data(wines)
som.wines <- som(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))
```

`unit.distances` *Calculate distances between units in a SOM*

Description

Calculate distances between units in a SOM.

Usage

```
unit.distances(grid, toroidal)
```

Arguments

`grid`	an object of class `somgrid`.
`toroidal`	if true, edges of the map are joined so that the topology is that of a torus.

Value

Returns a (symmetrical) matrix containing distances. When `grid$n.hood` equals "circular", Euclidean distances are used; for `grid$n.hood` is "square" maximum distances. If `toroidal` equals `TRUE`, maps are joined at the edges and distances are calculated for the shortest path.

Author(s)

Ron Wehrens

Examples

```
library(kohonen)
data(wines)

kohmap <- som(wines, grid = somgrid(5, 5, "hexagonal"), rlen=10)

dists <- unit.distances(kohmap$grid, toroidal=FALSE)
plot(kohmap, type="property", property=dists[1,],
     main="Distances to unit 1", zlim=c(0,6),
     palette = rainbow, ncolors = 7, contin = TRUE)
dists <- unit.distances(kohmap$grid, toroidal=TRUE)
```

```
plot(kohmap, type="property", property=dists[1,],
     main="Distances to unit 1 (toroidal)", zlim=c(0,6),
     palette = rainbow, ncolors = 7, contin = TRUE)

kohmap <- som(wines, grid = somgrid(5, 5), rlen=10)

dists <- unit.distances(kohmap$grid, toroidal=FALSE)
plot(kohmap, type="property", property=dists[1,],
     main="Distances to unit 1", zlim=c(0,4),
     palette = rainbow, ncolors = 5, contin = TRUE)
dists <- unit.distances(kohmap$grid, toroidal=TRUE)
plot(kohmap, type="property", property=dists[1,],
     main="Distances to unit 1 (toroidal)", zlim=c(0,4),
     palette = rainbow, ncolors = 5, contin = TRUE)
```

wines *Wine data*

Description

A data frame containing 177 rows and thirteen columns; object `vintages` contains the class labels. For compatibility with older versions of the package, variable `wine.classes` is retained, too.

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.

Usage

```
data(wines)
```

Source

http://kdd.ics.uci.edu

References

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

xyf *Supervised version of Kohonen's self-organising maps*

Description

Supervised version of self-organising maps for mapping high-dimensional spectra or patterns to 2D. The name stands for X-Y fused SOMs. One vector for each object is created by concatenating X and Y, and a SOM is trained in the usual way, with one exception: the distance of an object to a unit is the sum of separate distances for X and Y spaces. Prediction is done only using the X-space. For continuous Y, the Euclidean distance is used; for categorical Y the Tanimoto distance.

Usage

```
xyf(data, Y, grid=somgrid(), rlen = 100, alpha = c(0.05, 0.01),
    radius = quantile(nhbrdist, 0.67) * c(1, -1), xweight = 0.5,
    contin, toroidal = FALSE, n.hood, keep.data = TRUE)
```

Arguments

`data`	a matrix, with each row representing an object.
`Y`	property that is to be modelled. In case of classification, Y is a matrix of zeros, with exactly one '1' in each row indicating the class. For prediction of continuous properties, Y is a vector. A combination is possible, too, but one then should take care of appropriate scaling.
`grid`	a grid for the representatives: 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 run from the given number to the negative value of that number; as soon as the neighbourhood gets smaller than one only the winning unit will be updated. The default is to start with a value that covers 2/3 of all unit-to-unit distances.
`xweight`	the weight given to the X map in the calculation of distances for updating Y. Default is 0.5.
`contin`	parameter indicating whether Y is continuous or categorical. The default is to check whether all row sums of Y equal 1: in that case `contin` is `FALSE`.
`toroidal`	if TRUE, the edges of the map are joined. Note that in a hexagonal toroidal map, the number of rows must be even.
`n.hood`	the shape of the neighbourhood, either "circular" or "square". The latter is the default for rectangular maps, the former for hexagonal maps.
`keep.data`	save data in return value.

Value

an object of class "kohonen" with components

`data`	data matrix, only returned if `keep.data == TRUE`.
`Y`	Y, only returned if `keep.data == TRUE`.
`contin`	parameter indicating whether Y is continuous or categorical.
`grid`	the grid, an object of class "somgrid".
`codes`	list of two matrices, containing codebook vectors for X and Y, respectively.
`changes`	matrix containing two columns of mean average deviations from code vectors. Column 1 contains deviations used for updating Y; column 2 for updating X.
`toroidal`	whether a toroidal map is used.
`unit.classif`	winning units for all data objects, only returned if `keep.data == TRUE`.
`distances`	distances of objects to their corresponding winning unit, only returned if `keep.data == TRUE`.
`method`	the type of som, here "xyf".

Author(s)

Ron Wehrens

References

W.J. Melssen, R. Wehrens, and L.M.C. Buydens. Chemom. Intell. Lab. Syst., 83, 99-113 (2006).

See Also

`som`, `bdk`, `plot.kohonen`, `predict.kohonen`

Examples

```
### Wine example
data(wines)
set.seed(7)

training <- sample(nrow(wines), 120)
Xtraining <- scale(wines[training,])
Xtest <- scale(wines[-training,],
               center = attr(Xtraining, "scaled:center"),
               scale = attr(Xtraining, "scaled:scale"))

xyf.wines <- xyf(Xtraining,
                 factor(wine.classes[training]),
                 grid = somgrid(5, 5, "hexagonal"))

xyf.prediction <- predict(xyf.wines, newdata=Xtest)
table(wine.classes[-training], xyf.prediction$prediction)
```

`yeast` *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.

Usage

```
data(yeast)
```

References

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

Index