Package 'kohonen'

March 21, 2017

Version 3.0.2

Title Supervised and Unsupervised Self-Organising Maps

Author Ron Wehrens and Johannes Kruisselbrink

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

Description Functions to train self-organising maps (SOMs). Also interrogation of the maps and prediction using trained maps are supported. The name of the package refers to Teuvo Kohonen, the inventor of the SOM.

License GPL (>= 2)

Depends R (>= 2.6.0)

Imports MASS, Rcpp (>= 0.12.7)

LinkingTo Rcpp

NeedsCompilation yes

Repository CRAN

Date/Publication 2017-03-21 06:58:55 UTC

R topics documented:

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

Value

Returns a numerical vector with the indices of the selected layers. An invalid selection leads to an error.

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 a factor. 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 a factor; if it is a vector of integer values, it will be converted to a factor.
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 factor.

Author(s)

Ron Wehrens

See Also

`som`,`xyf`,`supersom`

Examples

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

---

degelder *Powder pattern data by Rene de Gelder*

Description

X-ray powder patterns of 131 crystallographic structures, contributed by Rene de Gelder.

Usage

```
data(degelder)
```

Format

This yields a list with three components: the first component, '"patterns"', is a matrix of 131 rows and 441 variables, containing the powder patterns; the second component is "thetas", the 2theta values at which intensities have been measured. The final component, '"properties"', gives information on the crystallographic properties of the structures.

Source

Rene de Gelder, Institute of Molecules and Materials, Radboud University Nijmegen.

Examples

```
## Not run:
data(degelder)
set.seed(1)
geldermap <- wccsom(degelder$patterns, somgrid(6, 6, "hexagonal"))
options(digits = 3)
summary(geldermap, "unit", nr=1, properties = degelder$properties)

## End(Not run)
```

---

expandMap *Expand a self-organising map*

Description

Double the size of a map, imputing the codebookvectors of the new units by averiging their immediate neighbours.

Usage

```
expandMap(kohobj)
```

Arguments

kohobj	Object of class "kohonen")

Value

A new kohonen object, with a double size.

Author(s)

Ron Wehrens

Examples

```
data(yeast)
yeast.supersom <- supersom(yeast, somgrid(4, 4, "hexagonal"),
                           whatmap = 3:6, maxNA.fraction = .5)
yeast.supersom2 <- expandMap(yeast.supersom)
yeast.supersom3 <- supersom(yeast, yeast.supersom2$grid,
                            whatmap = 3:6, maxNA.fraction = .5,
                            init = yeast.supersom2$codes[3:6])
```

getCodes *Extract codebook vectors from a kohonen object*

Description

Utility function for extracting codebook vectors. These are present as a list element in a kohonen object, and themselves are a list as well, with one entry for each data layer. This function returns either a list of codebook matrices (if more layers are selected), or just one matrix (if one layer is selected).

Usage

```
getCodes(x, idx = 1:length(codes))
```

Arguments

x	An object of class kohonen.
idx	Indices of the layer(s) for which codebook vectors are returned.

Value

If idx is a single number, a matrix of codebook vectors; if it is a vector of numbers, a list of codebook matrices.

Author(s)

Ron Wehrens

See Also

supersom

Examples

```
data(wines)
set.seed(7)
som.wines <- som(scale(wines), grid = somgrid(5, 5, "hexagonal"))
dim(getCodes(som.wines))
```

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, user.weights = NULL,
                      maxNA.fraction = NULL, ...)
```

Arguments

x	An object of class kohonen.
newdata	Data matrix, with rows corresponding to objects.
whatmap, user.weights, 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 supersom for more information.
...	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 of the objects to the closest units. Distance measures are the same ones used in training the map.
whatmap,weights	Values used for these arguments.

Author(s)

Ron Wehrens

See Also

predict.kohonen, supersom

Examples

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

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

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(X = nir$spectra[nir$training,],
              Y = nir$composition[nir$training,],
              user.weights = c(3,1),
              grid = somgrid(6, 6, "hexagonal"), rlen=500)
plot(nirnet, "counts", main="Counts")

## Focus on compound 2 (water):
par(mfrow = c(1,2))
set.seed(13)
nirnet <- xyf(X = nir$spectra[nir$training,],
              Y = nir$composition[nir$training, 2, drop = FALSE],
              grid = somgrid(6, 6, "hexagonal"), rlen=500)
water.xyf <-
  predict(nirnet, newdata = nir$spectra[nir$training,],
          unit.predictions = getCodes(nirnet, 2),
          whatmap = 1)$prediction
plot(nirnet, "property", property = water.xyf[[1]],
     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(X = nir$spectra[nir$training,],
               Y = matrix(nir$temperature[nir$training], ncol = 1),
               user.weights = c(1,3),
               grid = somgrid(6, 6, "hexagonal"), rlen=500)
temp.xyf <- predict(nirnet2, newdata = nir$spectra[nir$training,],
                    unit.predictions = getCodes(nirnet2, 2),
                    whatmap = 1)$prediction

plot(nirnet2, "property", property = temp.xyf[[1]],
     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)
```

`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.kohonen` using the `type = "dist.neighbours" argument.`

Usage

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

Arguments

`kohobj`	An object of class `kohonen`.
`type`	Whether to calculate distances between the data objects, or the codebook vectors.
`whatmap`	What data layers to use. If unspecified the data layers defined in the kohonen object are used.

Value

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

Author(s)

Ron Wehrens

References

R. Wehrens and J. Kruisselbrink, in preparation.

See Also

unit.distances, supersom

Examples

```
data(wines)
set.seed(7)
sommap <- supersom(list(measurements = scale(wines),
                        vintages = vintages),
                   grid = somgrid(6, 4, "hexagonal"))
obj.dists <- object.distances(sommap, type = "data")
code.dists <- object.distances(sommap, type = "codes")
```

---

plot.kohonen *Plot kohonen object*

Description

Plot objects of class 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"),
                        whatmap = NULL, classif = NULL, labels = NULL,
                        pchs = NULL, main = NULL, palette.name = NULL,
                        ncolors, bgcol = NULL, zlim = NULL,
                        heatkey = TRUE, property, codeRendering = NULL,
                        keepMargins = FALSE, heatkeywidth = .2,
                        shape = c("round", "straight"), border = "black",
                        ...)
## S3 method for class 'kohonen'
identify(x, ...)
add.cluster.boundaries(x, clustering, lwd = 5, ...)
```

Arguments

x	kohonen object.
type	type of plot. (Wow!)
whatmap	For a "codes" plot: what maps to show; for the "dist.neighbours" plot: what maps to take into account when calculating distances to neighbouring units.
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.
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.
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.

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).

Author(s)

Ron Wehrens

See Also

`som`, `supersom`, `xyf`, `predict.kohonen`

Examples

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

kohmap <- xyf(scale(wines), vintages,
              grid = somgrid(5, 5, "hexagonal"), rlen=100)
plot(kohmap, type="changes")
counts <- plot(kohmap, type="counts", shape = "straight")

## show both sets of codebook vectors in the map
par(mfrow = c(1,2))
plot(kohmap, type="codes", main = c("Codes X", "Codes Y"))

par(mfrow = c(1,1))
similarities <- plot(kohmap, type="quality", palette.name = terrain.colors)
```

```
plot(kohmap, type="mapping",
     labels = as.integer(vintages), col = as.integer(vintages),
     main = "mapping plot")

## add background colors to units according to their predicted class labels
xyfpredictions <- classmat2classvec(getCodes(kohmap, 2))
bgcols <- c("gray", "pink", "lightgreen")
plot(kohmap, type="mapping", col = as.integer(vintages),
     pchs = as.integer(vintages), bgcol = bgcols[as.integer(xyfpredictions)],
     main = "another mapping plot", shape = "straight", border = NA)

## Show 'component planes'
set.seed(7)
sommap <- som(scale(wines), grid = somgrid(6, 4, "hexagonal"))
plot(sommap, type = "property", property = getCodes(sommap, 1)[,1],
     main = colnames(getCodes(sommap, 1))[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 <- 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(object.distances(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 desired property associated with the corresponding winning unit. These properties may be provided explicitly (argument `unit.predictions`) or implicitly (by providing `trainingdata`, that will be mapped to the SOM - the averages of the winning units for the trainingdata then will be used as unit.predictions). If not given at all, the codebook vectors of the map will be used.

Usage

```
## S3 method for class 'kohonen'
predict(object,
                          newdata = NULL,
                          unit.predictions = NULL,
                          trainingdata = NULL,
```

```
                    whatmap = NULL,
                    threshold = 0, ...)
```

Arguments

`object`	Trained network, containing one or more information layers.
`newdata`	List of data matrices, or one single data matrix, for which predictions are to be made. The data layers should match those in the trained map. If not presented, the training data in the map will be used.
`unit.predictions`	Explicit definition of the predictions for each unit. Should be a list of matrices, vectors or factors, of the same length as `object$codes`.
`trainingdata`	List of data matrices, or one single data matrix, determining the mapping of the training data. Normally, data stored in the `kohonen` object will be used for this, but one can also specify this argument explicitly. Layers should match the trained map.
`whatmap`	What layers to use in the mapping. If not provided the intersection of the names of the newdata, the trainingdata, and the codebook vectors in the `kohonen` object will be used. If no names are present in the data layers, or if whatmap is in numerical form, the data layers in the different data objects should match exactly.
`threshold`	Used in converting class predictions back into factors; see `classmat2classvec`.
`...`	Further arguments to be passed to `map.kohonen`, in particular `maxNA.fraction` and `weights`. If not provided they will be taken from `object`.

Details

The new data are mapped to the trained SOM using the layers indicated by the `whatmap` argument. The predictions correspond to the `unit.predictions`, normally corresponding to the averages of the training data mapping to individual units. If no `unit.predictions` are provided, the `trainingdata` will be used to calculate them - if `trainingdata` is not provided by the user and the `kohonen` object contains data, these will be used. If no objects of the training data are mapping to a particular unit, the prediction for that unit will be NA.

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`	vector of unit numbers to which objects in the newdata object are mapped.
`unit.predictions`	prediction values associated with map units. Again, when multiple properties are predicted, this is a list.
`whatmap`	the numbers of the data layers in the kohonen object used in the mapping on which the predictions are based.

Author(s)

Ron Wehrens

See Also

som,xyf, supersom, map

Examples

```
data(wines)

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

mygrid = somgrid(5, 5, "hexagonal")
som.wines <- supersom(trainingdata, grid = mygrid)

## ####################################################################
## Situation 0: obtain expected values for training data (all layers,
## also if not used in training) on the basis of the position in the map
som.prediction <- predict(som.wines)

## ####################################################################
## Situation 1: obtain predictions for all layers used in training

som.prediction <- predict(som.wines, newdata = testdata)
table(vintages[-training], som.prediction$predictions[["vintages"]])


## ####################################################################
## Situation 2: obtain predictions for the vintage based on the mapping
## of the sample characteristics only. There are several ways of doing this:

som.prediction <- predict(som.wines, newdata = testdata,
                          whatmap = "measurements")
table(vintages[-training], som.prediction$predictions[["vintages"]])

## same, but now indicated implicitly
som.prediction <- predict(som.wines, newdata = testdata[1])
table(vintages[-training], som.prediction$predictions[["vintages"]])

## if no names are present in the list elements whatmap needs to be
## given explicitly; note that the order of the data layers needs to be
## consistent with the kohonen object
som.prediction <- predict(som.wines, newdata = list(Xtest), whatmap = 1)
table(vintages[-training], som.prediction$predictions[["vintages"]])
```

```
## ############################################################
## Situation 3: predictions for layers not present in the original
## data. Training data need to be provided for those layers.
som.wines <- supersom(Xtraining, grid = mygrid)
som.prediction <- predict(som.wines, newdata = testdata,
                          trainingdata = trainingdata)
table(vintages[-training], som.prediction$predictions[["vintages"]])

## ############################################################
## yeast examples, including NA values

data(yeast)
training.indices <- sample(nrow(yeast$alpha), 300)
training <- rep(FALSE, nrow(yeast$alpha))
training[training.indices] <- TRUE

## unsupervised mapping, based on the alpha layer only. Prediction
## for all layers including alpha
yeast.som <- supersom(lapply(yeast, function(x) subset(x, training)),
                      somgrid(4, 6, "hexagonal"),
                      whatmap = "alpha", maxNA.fraction = .5)
yeast.som.prediction <-
  predict(yeast.som,
          newdata = lapply(yeast, function(x) subset(x, !training)))

table(yeast$class[!training], yeast.som.prediction$prediction[["class"]])

## ############################################################
## supervised mapping - creating the map is now based on both
## alpha and class, prediction for class based on the mapping of alpha.
yeast.som2 <- supersom(lapply(yeast, function(x) subset(x, training)),
                       grid = somgrid(4, 6, "hexagonal"),
                       whatmap = c("alpha", "class"), maxNA.fraction = .5)
yeast.som2.prediction <-
  predict(yeast.som2,
          newdata = lapply(yeast, function(x) subset(x, !training)),
          whatmap = "alpha")
table(yeast$class[!training], yeast.som2.prediction$prediction[["class"]])
```

---

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, supersom

Examples

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

---

supersom *Self- and super-organising maps*

Description

A supersom is an extension of self-organising maps (SOMs) to multiple data layers, possibly with different numbers and different types of variables (though equal numbers of objects). NAs are allowed. A weighted distance over all layers is calculated to determine the winning units during training. Functions som and xyf are simply wrappers for supersoms with one and two layers, respectively.

Usage

```
som(X, ...)
xyf(X, Y, ...)
supersom(data, grid=somgrid(), rlen = 100, alpha = c(0.05, 0.01),
         radius = quantile(nhbrdist, 0.67),
         whatmap = NULL, user.weights = 1, maxNA.fraction = 0L,
         keep.data = TRUE, dist.fcts = NULL,
         mode = c("online", "batch", "pbatch"), cores = -1, init,
         normalizeDataLayers = TRUE)
```

Arguments

X, Y	numerical data matrices, or factors.
data	list of data matrices (numerical) of factors. If a vector is entered, it will be converted to a one-column matrix.
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. Not used for the batch algorithm.
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.
whatmap	What data layers to use. If unspecified all layers are used.
user.weights	the weights given to individual layers. The default is to give all layers equal weight.
maxNA.fraction	the maximal fraction of values that may be NA to prevent the row or 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	vector of distance functions to be used for the individual data layers, of the same length as the data argument, or the same length of the whatmap argument. If the length of this vector is one, the same distance will be used for all layers. Admissable values currently are "sumofsquares", "euclidean", "manhattan", and "tanimoto". Default is to use "sumofsquares" for continuous data, and "tanimoto" for factors.
mode	type of learning algorithm.
cores	number of cores to use in the "pbatch" learning mode. The default, -1, corresponds to using all available cores.
init	list of matrices, initial values for the codebook vectors. The list should have the same length as the data list, and corresponding numbers of variables (columns). Each list element should have a number of rows corresponding to the number of units in the map.
normalizeDataLayers	boolean, indicating whether distance.weights should be calculated (see details section). If normalizeDataLayers == FALSE the user weights are applied to the data immediately.
...	Further arguments for the supersom function presented to the som or xyf wrappers.

Details

In order to avoid some layers to overwhelm others, simply because of the scale of the data points, the `supersom` function by default applies internal weights to balance this. The `user.weights` argument is applied on top of that: the result is that when a user specifies equal weights for all layers (the default), all layers contribute equally to the global distance measure. For large data sets (defined as containing more than 500 records), a sample of size 500 is used to calculate the mean distances in each data layer. If `normalizeDataLayers == FALSE` the user weights are applied directly to the data (`distance.weights` are set to 1).

Value

An object of class "kohonen" with components

`data`	data matrix, only returned if `keep.data == TRUE`.
`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.
`alpha, radius, user.weights, whatmap, maxNA.fraction`	input arguments presented to the function.
`distance.weights`	if `normalizeDataLayers` weights to equalize the influence of the individual data layers, else a vector of ones.
`dist.fcts`	distance functions corresponding to all layers of the data, not just the ones indicated by the whatmap argument.

Author(s)

Ron Wehrens and Johannes Kruisselbrink

References

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

See Also

`somgrid`, `plot.kohonen`, `predict.kohonen`, `map.kohonen`

Examples

```
data(wines)

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

```
## xyf
xyf.wines <- xyf(scale(wines), vintages, grid = somgrid(5, 5, "hexagonal"))
summary(xyf.wines)

## supersom example
data(yeast)
yeast.supersom <- supersom(yeast, somgrid(6, 6, "hexagonal"),
                           whatmap = c("alpha", "cdc15", "cdc28", "elu"),
                           maxNA.fraction = .5)

plot(yeast.supersom, "changes")

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")
```

`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 *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, 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.

Author(s)

Ron Wehrens

Examples

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

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)
```

---

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.

Usage

```
data(wines)
```

Source

http://kdd.ics.uci.edu

References

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

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