

Package: som.nn (via r-universe)

September 1, 2024

Type Package

Title Topological k-NN Classifier Based on Self-Organising Maps

Version 1.4.4

Author Andreas Dominik

Maintainer Andreas Dominik <andreas.dominik@mni.thm.de>

Encoding UTF-8

Imports hexbin, class, kohonen, som, methods, graphics, grDevices,
stats, utils

Description A topological version of k-NN: An abstract model is build
as 2-dimensional self-organising map. Samples of unknown class
are predicted by mapping them on the SOM and analysing class
membership of neurons in the neighbourhood.

License GPL-3

RoxygenNote 7.3.1

NeedsCompilation no

Date/Publication 2024-04-03 17:00:02 UTC

Repository <https://andreasdominik.r-universe.dev>

RemoteUrl <https://github.com/cran/som.nn>

RemoteRef HEAD

RemoteSha f5d6f4842d8da472228a0a9244e6b68343b7bb20

Contents

som.nn-package	2
dist.fun.bubble	3
dist.fun.inverse	3
dist.fun.linear	4
dist.fun.tricubic	5
dist.torus	5
initialize,SOMnn-method	6
norm.linear	8

norm.softmax	9
plot,SOMnn,ANY-method	9
predict,SOMnn-method	11
round.proBABILITIES	12
som.nn.accuracy	13
som.nn.all.accuracy	14
som.nn.confusion	14
som.nn.continue	15
som.nn.export.kohonen	17
som.nn.export.som	18
som.nn.multitrain	18
som.nn.set	21
som.nn.train	22
som.nn.validate	25
som.nn.visual	26
SOMnn-class	27
Index	29

som.nn-package	<i>Topological k-NN Classifier Based on Self-Organising Maps</i>
----------------	------------------------------------------------------------------

Description

The package `som.nn` provides tools to train self-organising maps and predict class memberships by means of a k-NN-like classifier.

Details

The functions `som.nn.train` and `som.nn.continue` are used to train and re-train self-organising maps. The training can be performed with functions of the packages `kohonen`, `som`, `class` or with pure-R-implementations with distance function `bubble` (kernel `internal`) or `gaussian` (kernel `gaussian`). (Remark: The pure-R-implementations actually are faster as the external calls to C implementations in the above-mentioned packages!).

In contrast to a normal som training, class labels are required for all training samples. These class labels are used to assign classes to the codebook vectors (i.e. the neurons of the map) after the training and build the set of reference vectors. This reference is used for nearest-neighbour classification.

The nearest neighbour classifier is implemented as `predict` method. It is controlled by the following parameters:

- `dist.fun`: the distance function to weight the distance of reference vectors and the sample to be predicted.
- `max.dist`: the maximum distance to be considered.

Some distance functions are provided in the package (`linear`, `bubble`, `inverse` and `tricubic`) but any custom function can be defined as well.

The prediction differs significantly from a standard nearest-neighbour classifier, because the neighbourhood is not defined by the distance between reference vectors and unknown sample vector. Instead the neighbourhood of the neurons on the self-organising map is used.

Because the som have been generated by an unsupervised training, the classifier is robust against overtraining.

In addition the abstract model can be visualised as 2-dimensional map, using the plot method.

dist.fun.bubble *Bubble distance functions for topological k-NN classifier*

Description

The function is used as distance-dependent weight w for k-NN voting.

Usage

```
dist.fun.bubble(x, sigma = 1.1)
```

Arguments

x	Distance or numeric vector or matrix of distances.
sigma	Maximum distance to be considered. Default is 1.1.

Details

The function returns 1.0 for $0 < x \leq \sigma$ and 0.0 for $x > \sigma$.

Value

Distance-dependent weight.

dist.fun.inverse *Inverse exponential distance functions for topological k-NN classifier*

Description

The function is used as distance-dependent weight w for k-NN voting.

Usage

```
dist.fun.inverse(x, sigma = 1.1)
```

Arguments

x	Distance or numeric vector or matrix of distances.
sigma	Maximum distance to be considered. Default is 1.1.

Details

The function returns 1.0 for $x = 0$, 0.0 for $x \geq \sigma$ and

$$1/(x + 1)^{1/\sigma}$$

for $0 < x < \sigma$.

Value

Distance-dependent weight.

 dist.fun.linear

Linear distance functions for topological k-NN classifier

Description

The function is used as distance-dependent weight w for k-NN voting.

Usage

```
dist.fun.linear(x, sigma = 1.1)
```

Arguments

`x` Distance or numeric vector of distances.
`sigma` Maximum distance to be considered. Default is 1.1.

Details

The function returns 1.0 for $x = 0$, 0.0 for $x \geq \sigma$ and

$$1 - x/\sigma$$

for $0 < x < \sigma$.

Value

Distance-dependent weight.

dist.fun.tricubic	<i>Tricubic distance functions for topological k-NN classifier</i>
-------------------	--------------------------------------------------------------------

Description

The tricubic function is used as distance-dependent weight w for k-NN voting.

Usage

```
dist.fun.tricubic(x, sigma = 1)
```

Arguments

x	Distance or numeric vector or matrix of distances.
sigma	Maximum distance to be considered.

Details

The function returns 1.0 for $x = 0$, 0.0 for $x \geq \sigma$ and

$$w(x) = (1 - x^3/\sigma^3)^3$$

for $0 < x < \sigma$.

Value

Distance-dependent weight.

dist.torus	<i>Torus distance matrix</i>
------------	------------------------------

Description

Calculates the distance matrix of points on the surface of a torus.

Usage

```
dist.torus(coors)
```

Arguments

coors	data.frame or matrix with two columns with x- and y-coordinates.
-------	------------------------------------------------------------------

Details

A rectangular plane is considered as torus (i.e. on an endless plane that continues on the left, when leaving at the right side, and in the same way connects top and bottom border). Distances between two points on the plane are calculated as the shortest distance between the points on the torus surface.

Value

Complete distance matrix with diagonal and upper triangle values.

initialize,SOMnn-method

Constructor of SOMnn Class

Description

The constructor creates a new object of type SOMnn.

Usage

```
## S4 method for signature 'SOMnn'
initialize(
  .Object,
  name,
  codes,
  qerror,
  class.idx,
  classes,
  class.counts,
  class.freqs,
  confusion,
  measures,
  accuracy,
  xdim,
  ydim,
  len.total,
  toroidal,
  norm,
  norm.center,
  norm.scale,
  dist.fun,
  max.dist,
  strict
)
```

Arguments

.Object	SOMnn object
name	optional name of the model.
codes	data.frame with codebook vectors of the som.
qerror	sum of the mapping errors of the training data.
class.idx	numeric index of column with categories.
classes	character vector with names of categories.
class.counts	data.frame with class hits for each neuron.
class.freqs	data.frame with class frequencies for each neuron (freqs sum up to 1).
confusion	data.frame with confusion matrix for training data.
measures	data.frame with classes as rows and the columns sensitivity, specificity and accuracy for each class.
accuracy	Overall accuracy.
xdim	number of neurons in x-direction of the som.
ydim	number of neurons in y-direction of the som.
len.total	total number of training steps, performed to create the model.
toroidal	logical; if TRUE, the map is toroidal (i.e. borderless).
norm	logical; if TRUE, data is normalised before training and mapping. Parameters for normalisation of training data is stored in the model and applied before mapping of test data.
norm.center	vector of centers for each column of training data.
norm.scale	vector of scale factors for each column of training data.
dist.fun	function; kernel for the kNN classifier.
max.dist	maximum distance σ for the kNN classifier.
strict	Minimum vote for the winner (if the winner's vote is smaller than strict, "unknown" is reported as class label (default = 0.8)).

Details

The constructor needs not to be called directly, because the normal way to create a SOMnn object is to use `som.nn.train`.

Examples

```
## Not run:
new.som <- new("SOMnn", name = name,
              codes = codes,
              qerror = qerror,
              classes = classes,
              class.idx = class.idx,
              class.counts = class.counts,
              class.freqs = class.freqs,
              confusion = confusion,
```

```
measures = measures,  
accuracy = accuracy,  
xdim = xdim,  
ydim = ydim,  
len.total = len.total,  
toroidal = toroidal,  
norm = norm,  
norm.center = norm.center,  
norm.scale = norm.scale,  
dist.fun = dist.fun,  
max.dist = max.dist.  
strict = strict)  
  
## End(Not run)
```

norm.linear

Linear normalisation

Description

Calculates a linear normalisation for the class frequencies.

Usage

```
norm.linear(x)
```

Arguments

x vector of votes for classes

Details

The function is applied to a vector to squeeze the values in a way that they sum up to 1.0:

```
som.nn.linnorm(x) = x / sum(x)
```

Linear normalisation is used to normalise class distribution during prediction. Results seems often more reasonable, compared to softmax. The S4 predict function for Class SOMnn allows to specify the normalisation function as parameter.

Value

Vector of normalised values.

 norm.softmax

Softmax normalisation

Description

Calculates a softmax-like normalisation for the class frequencies.

Usage

```
norm.softmax(x, t = 0.2)
```

Arguments

x	vector of votes for classes
t	temperature parameter.

Details

Softmax function is applied to a vector to squeeze the values in a way that they sum up to 1.0:

$$\text{som.nn.softmax}(x) = \exp(x/T) / \text{sum}(\exp(x/T))$$

Low values for T result in a strong separation of output values. High values for T make output values more equal.

Value

Vector of softmax normalised values.

 plot, SOMnn, ANY-method *Plot method for S4 class SOMnn*

Description

Creates a plot of the hexagonal som in the model of type SOMnn.

Usage

```
## S4 method for signature 'SOMnn,ANY'
plot(
  x,
  title = TRUE,
  col = NA,
  onlyDefCols = FALSE,
  edit.cols = FALSE,
  show.legend = TRUE,
  legend.loc = "bottomright",
```

```

legend.width = 4,
window.width = NA,
window.height = NA,
show.box = TRUE,
show.counter.border = 0.98,
predict = NULL,
add = FALSE,
pch.col = "black",
pch = 19,
...
)

```

Arguments

<code>x</code>	trained som of type SOMnn.
<code>title</code>	logical; if TRUE, slots name and date are used as main title.
<code>col</code>	defines colours for the classes of the dataset. Possible values include: NA: default value; colours are generated with <code>rainbow</code> , a vector of colour definitions or a <code>data.frame</code> with categories in the first and respective colours in the second column.
<code>onlyDefCols</code>	logical; if TRUE, only categories are plotted, for which colours are defined. Default: FALSE.
<code>edit.cols</code>	logical; if TRUE, colour definitions can be edited interactively before plotting. Default: FALSE.
<code>show.legend</code>	logical; if TRUE, a legend is displayed,. Default: TRUE.
<code>legend.loc</code>	Legend position as specified for legend . Default is "bottomright".
<code>legend.width</code>	size of the legend.
<code>window.width</code>	Manual setting of window width. Default is NA.
<code>window.height</code>	Manual setting of window height. Default is NA.
<code>show.box</code>	Show frame around the plot . Default is TRUE.
<code>show.counter.border</code>	Percentile as limit for the display of labels in the pie charts. Default is 0.98. Higher counts are displayed as numbers in the neuron.
<code>predict</code>	<code>data.frame</code> as returned by the <code>som.nn::predict</code> function or a <code>data.frame</code> or matrix that follows the specification: If columns <code>x</code> and <code>y</code> exist, these are used as coordinates for the target neuron; otherwise the first two columns are used. Default: NULL.
<code>add</code>	logical; if TRUE, points are plotted on an existing plot. This can be used to stepwise plot points of different classes with different colours or symbols.
<code>pch.col</code>	Colour of the markers for predicted samples.
<code>pch</code>	Symbol of the markers for predicted samples.
<code>...</code>	More parameters as well as general plot parameters are allowed; see par .

Details

In addition to the required parameters, many options can be specified to plot predicted samples and to modify colours, legend and scaling.

Examples

```
## get example data and add class labels:
data(iris)
species <- iris$Species

## train with default radius = diagonal / 2:
rlen <- 500
som <- som.nn.train(iris, class.col = "Species", kernel = "internal",
  xdim = 15, ydim = 9, alpha = 0.2, len = rlen,
  norm = TRUE, toroidal = FALSE)

## continue training with different alpha and radius;
som <- som.nn.continue(som, iris, alpha = 0.02, len=500, radius = 5)
som <- som.nn.continue(som, iris, alpha = 0.02, len=500, radius = 2)

## predict some samples:
unk <- iris[!(names(iris) %in% "Species")]

setosa <- unk[species=="setosa",]
setosa <- setosa[sample(nrow(setosa), 20),]

versicolor <- unk[species=="versicolor",]
versicolor <- versicolor[sample(nrow(versicolor), 20),]

virginica <- unk[species=="virginica",]
virginica <- virginica[sample(nrow(virginica), 20),]

p <- predict(som, unk)
head(p)

## plot:
plot(som)
dev.off()
plot(som, predict = predict(som, setosa))
plot(som, predict = predict(som, versicolor), add = TRUE, pch.col = "magenta", pch = 17)
plot(som, predict = predict(som, virginica), add = TRUE, pch.col = "white", pch = 8)
```

predict,SOMnn-method *predict method for S4 class SOMnn*

Description

Predicts categories for a table of data, based on the hexagonal som in the model. This S4 method is a wrapper for the predict method stored in the slot predict of a model of type SOMnn.

Usage

```
## S4 method for signature 'SOMnn'
predict(object, x)
```

Arguments

object	object of type SOMnn.
x	data.frame with rows of data to be predicted.

Details

The function returns the winner neuron in codes for each test vector in `x`. `x` is organised as one vector per row and must have the same number of columns (i.e. dimensions) and the identical column names as stored in the SOMnn object.

If data have been normalised during training, the same normalisation is applied to the unknown data to be predicted.

Probabilities are softmax normalised by default.

Value

```
\code{data.frame} with columns:
  \code{winner}, \code{x}, \code{y}, the predicted probabilities
  for all categories and the prediction
  as category index (column name \code{prediction}) and
  class label (column name \code{pred.class}).
```

round.probabilities *Advanced rounding of vectors*

Description

Rounds a vector of probabilities preserving their sum.

Usage

```
## S3 method for class 'probabilities'
round(x, digits = 2)
```

Arguments

x	numeric vector of values.
digits	demanded precision

Details

In general, if a vector of floating point values is rounded, the sum is not preserved. For a vector of probabilities (which sum up to 1.0), this may lead to strange results. This function rounds all values of the vector and takes care, that the sum is not changed (with a precision given in `digits`).

som.nn.accuracy	<i>Calculate accuracy measures</i>
-----------------	------------------------------------

Description

Calculates the sensitivity, specificity and overall accuracy for a prediction result if the corresponding vector of true class labels is provided.

Usage

```
som.nn.accuracy(x, class.labels)
```

Arguments

`x` data.frame with the predictions as returned by the SOM.nn predict method.
`class.labels` vector of correct class labels for the predictions.

Details

Sensitivity is the classifier's ability to correctly identify samples of a specific class A. It is defined as

$$sens_A = TP_A / (TP_A + FN_A)$$

with TP = true positives and FN = false negatives. This is equivalent to the ratio of (correctly identified samples of class A) / (total number of samples of class A).

Specificity is the classifier's ability to correctly identify samples not of a specific class A. It is defined as

$$spec_A = TN_A / (TN_A + FP_A)$$

with TN = true negatives and FP = false positives. This is equivalent to the ratio of (correctly identified samples not in class A) / (total number of samples not in class A).

Accuracy is the classifier's ability to correctly classify samples of a specific class A. It is defined as

$$acc_A = (TP_A + TN_A) / total$$

with TP = true positives, TN = true negatives and total = total number of samples of a class. This is equivalent to the ratio of (correctly classified samples) / (total number of samples).

Value

data.frame containing sensitivity, specificity and accuracy for all class labels in the data set.

som.nn.all.accuracy *Calculate overall accuracy*

Description

Calculates the accuracy over all class labels for a prediction result if the corresponding vector of true class labels is provided.

Usage

```
som.nn.all.accuracy(x, class.labels)
```

Arguments

x data.frame with the predictions as returned by the SOM.nn predict method.
class.labels vector of correct class labels for the predictions.

Details

It is defined as

$$acc = (TP + TN)/total = sum(diag(cmat))/sum(cmat)$$

with TP = true positives, TN = true negatives and total = total number of samples of a class. This is equivalent to the ratio of (correctly classified samples) / (total number of samples).

Value

one value overall accuracy.

som.nn.confusion *Calculate confusion matrix*

Description

Calculates the confusion matrix for a prediction result if the corresponding vector of true class labels is provided.

Usage

```
som.nn.confusion(x, class.labels)
```

Arguments

x data.frame with the predictions as returned by the SOM.nn predict method.
class.labels vector of correct class labels for the predictions.

Details

The confusion matrix (also called table of confusion) displays the number of predicted class labels for each actual class. Example:

	pred. cat	pred. dog	pred. rabbit	unknown
actual cat	5	3	0	0
actual dog	2	3	1	0
actual rabbit	0	2	9	2

The confusion matrix includes a column unknown displaying the samples for which no unambiguous prediction is possible.

Value

data.frame containing the confusion matrix.

som.nn.continue	<i>Continue hexagonal som training</i>
-----------------	----------------------------------------

Description

An existing self-organising map with hexagonal topology is further trained and a model created for prediction of unknown samples. In contrast to a "normal" som, class-labels for all samples of the training set are required to build the model.

Usage

```
som.nn.continue(
  model,
  x,
  kernel = "internal",
  len = 0,
  alpha = 0.2,
  radius = 0
)
```

Arguments

model	model of type SOMnn.
x	data.frame with training data. Samples are requested as rows and taken randomly for the training steps. All columns except of the class labels are considered to be attributes and parts of the training vector. x must include the same columns as the data.frame with which the model have been trained originally. One column is needed as class labels. The column with class labels is selected by the slot class.idx of the model.

kernel	Kernel for som training. One of the predefined kernels "bubble" and "gaussian" == train with the R-implementation or "SOM" == train with <code>SOM</code> or "kohonen" == train with <code>som</code> (kohonen: : som) or "som" == train with <code>som</code> (som: : som). If a function is specified (as closure, not as character) the specified custom function is used for training.
len	number of steps to be trained (steps - not epochs!).
alpha	initial training rate; default 0.02.
radius	inital radius for SOM training. If Gaussian distance function is used, radius corresponds to sigma.

Details

Any specified custom kernel function is used for som training. The function must match the signature `kernel(data, grid, rlen, alpha, radius, init, toroidal)`, with arguments:

- data numeric matrix of training data; one sample per row
- classes: optional character vector of classes for training data
- grid somgrid, generated with `somgrid`
- rlen number of training steps
- alpha training rate
- radius training radius
- init numeric matrix of initial codebook vectors; one code per row
- toroidal logical; TRUE, if the topology of grid is toroidal

The returned value must be a list with at minimum one element

- codes: numeric matrix of result codebook vectors; one code per row

Value

S4 object of type `\code{\link{SOMnn}}` with the trained model

Examples

```
## get example data and add class labels:
data(iris)
species <- iris$Species

## train with default radius = diagonal / 2:
rlen <- 500
som <- som.nn.train(iris, class.col = "Species", kernel = "internal",
                  xdim = 15, ydim = 9, alpha = 0.2, len = rlen,
                  norm = TRUE, toroidal = FALSE)

## continue training with different alpha and radius;
som <- som.nn.continue(som, iris, alpha = 0.02, len=500, radius = 5)
som <- som.nn.continue(som, iris, alpha = 0.02, len=500, radius = 2)
```



```
## predict some samples:
unk <- iris[!(names(iris) %in% "Species")]

setosa <- unk[species=="setosa",]
setosa <- setosa[sample(nrow(setosa), 20),]

versicolor <- unk[species=="versicolor",]
versicolor <- versicolor[sample(nrow(versicolor), 20),]

virginica <- unk[species=="virginica",]
virginica <- virginica[sample(nrow(virginica), 20),]

p <- predict(som, unk)
head(p)

## plot:
plot(som)
dev.off()
plot(som, predict = predict(som, setosa))
plot(som, predict = predict(som, versicolor), add = TRUE, pch.col = "magenta", pch = 17)
plot(som, predict = predict(som, virginica), add = TRUE, pch.col = "white", pch = 8)
```

som.nn.export.kohonen *Export a som.nn model as object of type kohonen*

Description

An existing model of type SOMnn is exported as object of type kohonen for use with the tools of the package kohonen.

Usage

```
som.nn.export.kohonen(model, train)
```

Arguments

model	model of type SOMnn.
train	training data

Details

Training data is necessary to generate the kohonen object.

Value

Vist of type `kohonen` with the trained som.
See `\link[kohonen]{som}` for details.

som.nn.export.som *Export a som.nn model as object of type SOM*

Description

An existing model of type SOMnn is exported as object of type SOM for use with the tools of the package class.

Usage

```
som.nn.export.som(model)
```

Arguments

model model of type SOMnn.

Value

List of type `SOM` with the trained som.
See `\link[class]{SOM}` for details.

som.nn.multitrain *Multi-step hexagonal som training*

Description

A self-organising map with hexagonal topology is trained in several steps and a model of Type SOMnn created for prediction of unknown samples. In contrast to a "normal" som, class-labels for all samples of the training set are required to build the topological model after SOM training.

Usage

```
som.nn.multitrain(
  x,
  class.col = 1,
  kernel = "internal",
  xdim = 7,
  ydim = 5,
  toroidal = FALSE,
  len = c(0),
  alpha = c(0.2),
  radius = c(0),
  focus = 1,
  norm = TRUE,
  dist.fun = dist.fun.inverse,
  max.dist = 1.1,
  name = "som.nn job"
)
```

Arguments

<code>x</code>	data.frame with training data. Samples are requested as rows and taken randomly for the training steps. All columns except of the class labels are considered to be attributes and parts of the training vector. One column is needed as class labels. The column with class labels is selected by the argument <code>class.col</code> .
<code>class.col</code>	single string or number. If class is a string, it is considered to be the name of the column with class labels. If class is a number, the respective column will be used as class labels (after being coerced to character). Default is 1.
<code>kernel</code>	kernel for som training. One of the predefined kernels "bubble": train with the R-implementation or "gaussian": train with the R-implementation of the Gaussian kernel or "SOM": train with <code>SOM</code> (<code>class::SOM</code>) or "kohonen": train with <code>som</code> (<code>kohonen::som</code>) or "som": train with <code>som</code> (<code>som::som</code>). If a function is specified (as closure, not as character) the specified custom function is used for training.
<code>xdim</code>	dimension in x-direction.
<code>ydim</code>	dimension in y-direction.
<code>toroidal</code>	logical; if TRUE an endless som is trained as on the surface of a torus. default: FALSE.
<code>len</code>	vector of numberis of steps to be trained (steps - not epochs!). the length of len defines the number of training rounds to be performed.
<code>alpha</code>	initial training rate; the learning rate is decreased linearly to 0.0 for the last training step. Default: 0.02. If <code>length(alpha) > 1</code> , the length must be the same as for len and defines different alphas for each training round.
<code>radius</code>	initial radius for SOM training. If Gaussian distance function is used, radius corresponds to sigma. The distance is decreased linearly to 1.0 for the last training step. If <code>radius = 0</code> (default), the diameter of the SOM is used as initial radius. If <code>length(radius) > 1</code> , the length must be the same as for len and defines different radii for each training round.
<code>focus</code>	Enhancement factor for focussing of training of "dirty" samples.
<code>norm</code>	logical; if TRUE, input data is normalised by <code>scale(x, TRUE, TRUE)</code> .
<code>dist.fun</code>	parameter for k-NN prediction: Function used to calculate distance-dependent weights. Any distance function must accept the two parameters <code>x</code> (distance) and <code>sigma</code> (maximum distance to give a weight > 0.0). Default is <code>dist.fun.inverse</code> .
<code>max.dist</code>	parameter for k-NN prediction: Parameter <code>sigma</code> for <code>dist.fun</code> . Default is 2.1. In order to avoid rounding issues, it is recommended not to use exact integers as limit, but values like 1.1 to make sure, that all neurons within distance 1 are included.
<code>name</code>	optional name for the model. Name will be stored as slot <code>model@name</code> in the trained model.

Details

Besides of the predefined kernels "bubble", "gaussian", "SOM", "kohonen" or "som", any specified custom kernel function can be used for som training. The function must match the signature `kernel(data, grid, rlen, alpha, radius, init, toroidal)`, with arguments:

- data: numeric matrix of training data; one sample per row
- classes: optional character vector of classes for training data
- grid: somgrid, generated with `somgrid`
- rlen: number of training steps
- alpha: training rate
- radius: training radius
- init: numeric matrix of initial codebook vectors; one code per row
- toroidal: logical; TRUE, if the topology of grid is toroidal

The returned value must be a list with at minimum one element

- codes: numeric matrix of result codebook vectors; one code per row

If `focus > 1` enhancement of dirty samples is activated: Training samples, mapped to neuron with `>1` classes, are preferred in the next training step.

Value

S4 object of type `\code{\link{SOMnn}}` with the trained model

Examples

```
## get example data and add class labels:
data(iris)
species <- iris$Species

## train with default radius = diagonal / 2:
rlen <- 500
som <- som.nn.train(iris, class.col = "Species", kernel = "internal",
                  xdim = 15, ydim = 9, alpha = 0.2, len = rlen,
                  norm = TRUE, toroidal = FALSE)

## continue training with different alpha and radius;
som <- som.nn.continue(som, iris, alpha = 0.02, len=500, radius = 5)
som <- som.nn.continue(som, iris, alpha = 0.02, len=500, radius = 2)

## predict some samples:
unk <- iris[!(names(iris) %in% "Species")]

setosa <- unk[species=="setosa",]
setosa <- setosa[sample(nrow(setosa), 20),]

versicolor <- unk[species=="versicolor",]
versicolor <- versicolor[sample(nrow(versicolor), 20),]

virginica <- unk[species=="virginica",]
virginica <- virginica[sample(nrow(virginica), 20),]

p <- predict(som, unk)
```

```

head(p)

## plot:
plot(som)
dev.off()
plot(som, predict = predict(som, setosa))
plot(som, predict = predict(som, versicolor), add = TRUE, pch.col = "magenta", pch = 17)
plot(som, predict = predict(som, virginica), add = TRUE, pch.col = "white", pch = 8)

```

som.nn.set

Set parameters for k-NN-like classifier in som.nn model

Description

Parameters for the k-NN-like classification can be set for an existing model of type SOMnn after training.

Usage

```

som.nn.set(
  model,
  x,
  dist.fun = NULL,
  max.dist = NULL,
  strict = NULL,
  name = NULL
)

```

Arguments

model	model of type SOMnn.
x	data.frame with training data. Samples are requested as rows and taken randomly for the training steps. All columns except of the class labels are considered to be attributes and parts of the training vector. x must include the same columns as the data.frame with which the model have been trained originally. One column is needed as class labels. The column with class labels is selected by the slot <code>class.idx</code> of the model.
dist.fun	distance function for weighting distances between codebook vectors on the som (kernel for k-NN classifier).
max.dist	maximum distance to be considered by the nearest-neighbour counting.
strict	strictness for class label assignment. Default = 0.8.
name	new name of the model.

Details

The distance function defines the behaviour of the k-nearest-neighbour algorithm. Choices for the distance function include `dist.fun.inverse` or `dist.fun.tricubic`, as defined in this package, or any other function that accepts exactly two arguments `x` (the distance) and `sigma` (a parameter defined by `max.distance`).

A data set must be presented to calculate the accuracy statistics of the modified predictor.

Value

S4 object of type `\link{SOMnn}` with the updated model.

See Also

[dist.fun.bubble](#), [dist.fun.linear](#), [dist.fun.inverse](#), [dist.fun.tricubic](#).

som.nn.train

Hexagonal som training

Description

A self-organising map with hexagonal topology is trained and a model of Type `SOMnn` created for prediction of unknown samples. In contrast to a "normal" som, class-labels for all samples of the training set are required to build the topological model after SOM training.

Usage

```
som.nn.train(  
  x,  
  class.col = 1,  
  kernel = "internal",  
  xdim = 7,  
  ydim = 5,  
  toroidal = FALSE,  
  len = 0,  
  alpha = 0.2,  
  radius = 0,  
  norm = TRUE,  
  dist.fun = dist.fun.inverse,  
  max.dist = 1.1,  
  strict = 0.8,  
  name = "som.nn job"  
)
```

Arguments

<code>x</code>	data.frame with training data. Samples are requested as rows and taken randomly for the training steps. All columns except of the class labels are considered to be attributes and parts of the training vector. One column is needed as class labels. The column with class labels is selected by the argument <code>class.col</code> .
<code>class.col</code>	single string or number. If class is a string, it is considered to be the name of the column with class labels. If class is a number, the respective column will be used as class labels (after being coerced to character). Default is 1.
<code>kernel</code>	kernel for som training. One of the predefined kernels "bubble": train with the R-implementation or "gaussian": train with the R-implementation of the Gaussian kernel or "SOM": train with <code>SOM(class::SOM)</code> or "kohonen": train with <code>som(kohonen::som)</code> or "som": train with <code>som(som::som)</code> . If a function is specified (as closure, not as character) the specified custom function is used for training.
<code>xdim</code>	dimension in x-direction.
<code>ydim</code>	dimension in y-direction.
<code>toroidal</code>	logical; if TRUE an endless som is trained as on the surface of a torus. default: FALSE.
<code>len</code>	number of steps to be trained (steps - not epochs!).
<code>alpha</code>	initial training rate; the learning rate is decreased linearly to 0.0 for the last training step. Default: 0.02.
<code>radius</code>	initial radius for SOM training. If Gaussian distance function is used, radius corresponds to sigma. The distance is decreased linearly to 1.0 for the last training step. If <code>radius = 0</code> (default), the diameter of the SOM is used as initial radius.
<code>norm</code>	logical; if TRUE, input data is normalised by <code>scale(x, TRUE, TRUE)</code> .
<code>dist.fun</code>	parameter for k-NN prediction: Function used to calculate distance-dependent weights. Any distance function must accept the two parameters <code>x</code> (distance) and <code>sigma</code> (maximum distance to give a weight > 0.0). Default is <code>dist.fun.inverse</code> .
<code>max.dist</code>	parameter for k-NN prediction: Parameter <code>sigma</code> for <code>dist.fun</code> . Default is 2.1. In order to avoid rounding issues, it is recommended not to use exact integers as limit, but values like 1.1 to make sure, that all neurons within distance 1 are included.
<code>strict</code>	Minimum vote for the winner (if the winner's vote is smaller than <code>strict</code> , "unknown" is reported as class label (default = 0.8)).
<code>name</code>	optional name for the model. Name will be stored as slot <code>model@name</code> in the trained model.

Details

Besides of the predefined kernels "internal", "gaussian", "SOM", "kohonen" or "som", any specified custom kernel function can be used for som training. The function must match the signature `kernel(data, grid, rlen, alpha, radius, init, toroidal)`, with arguments:

- `data`: numeric matrix of training data; one sample per row

- `classes`: optional character vector of classes for training data
- `grid`: somgrid, generated with `somgrid`
- `rlen`: number of training steps
- `alpha`: training rate
- `radius`: training radius
- `init`: numeric matrix of initial codebook vectors; one code per row
- `toroidal`: logical; TRUE, if the topology of grid is toroidal

The returned value must be a list with at minimum one element

- `codes`: numeric matrix of result codebook vectors; one code per row

Value

S4 object of type `\link{SOMnn}` with the trained model

Examples

```
## get example data and add class labels:
data(iris)
species <- iris$Species

## train with default radius = diagonal / 2:
rlen <- 500
som <- som.nn.train(iris, class.col = "Species", kernel = "internal",
                  xdim = 15, ydim = 9, alpha = 0.2, len = rlen,
                  norm = TRUE, toroidal = FALSE)

## continue training with different alpha and radius;
som <- som.nn.continue(som, iris, alpha = 0.02, len=500, radius = 5)
som <- som.nn.continue(som, iris, alpha = 0.02, len=500, radius = 2)

## predict some samples:
unk <- iris[!(names(iris) %in% "Species")]

setosa <- unk[species=="setosa",]
setosa <- setosa[sample(nrow(setosa), 20),]

versicolor <- unk[species=="versicolor",]
versicolor <- versicolor[sample(nrow(versicolor), 20),]

virginica <- unk[species=="virginica",]
virginica <- virginica[sample(nrow(virginica), 20),]

p <- predict(som, unk)
head(p)

## plot:
plot(som)
dev.off()
```



```
plot(som, predict = predict(som, setosa))
plot(som, predict = predict(som, versicolor), add = TRUE, pch.col = "magenta", pch = 17)
plot(som, predict = predict(som, virginica), add = TRUE, pch.col = "white", pch = 8)
```

som.nn.validate *Predict class labels for a validation dataset*

Description

A model of type SOMnn is tested with a validation dataset. The dataset must include a column with correct class labels. The model is used to predict class labels. Confusion table, specificity, sensitivity and accuracy for each class are calculated.

Usage

```
som.nn.validate(model, x)
```

Arguments

model	model of type SOMnn.
x	data.frame with validation data. Samples are requested as rows. x must include the same columns as the data.frame with which the model have been trained originally. A column with correct class labels is needed. The column with class labels is selected by the slot class.idx of the model.

Details

Parameters stored in the model are applied for k-NN-like prediction. If necessary the parameters can be changed by [som.nn.set](#) before testing.

The function is only a wrapper and actually calls som.nn.continue with the test data and without training (i.e. len = 0).

Value

S4 object of type `\link{SOMnn}` with the unchanged model and the test statistics for the test data.

Examples

```
## get example data and add class labels:
data(iris)
species <- iris$Species

## train with default radius = diagonal / 2:
rlen <- 500
som <- som.nn.train(iris, class.col = "Species", kernel = "internal",
                   xdim = 15, ydim = 9, alpha = 0.2, len = rlen,
```

```

norm = TRUE, toroidal = FALSE)

## continue training with different alpha and radius;
som <- som.nn.continue(som, iris, alpha = 0.02, len=500, radius = 5)
som <- som.nn.continue(som, iris, alpha = 0.02, len=500, radius = 2)

## predict some samples:
unk <- iris[,!(names(iris) %in% "Species")]

setosa <- unk[species=="setosa",]
setosa <- setosa[sample(nrow(setosa), 20),]

versicolor <- unk[species=="versicolor",]
versicolor <- versicolor[sample(nrow(versicolor), 20),]

virginica <- unk[species=="virginica",]
virginica <- virginica[sample(nrow(virginica), 20),]

p <- predict(som, unk)
head(p)

## plot:
plot(som)
dev.off()
plot(som, predict = predict(som, setosa))
plot(som, predict = predict(som, versicolor), add = TRUE, pch.col = "magenta", pch = 17)
plot(som, predict = predict(som, virginica), add = TRUE, pch.col = "white", pch = 8)

```

som.nn.visual

Mapping function for SOMnn

Description

Maps a sample of unknown category to a self-organising map (SOM) stored in a object of type SOMnn.

Usage

```
som.nn.visual(codes, data)
```

Arguments

codes	data.frame with codebook vectors.
data	data.frame with data to be mapped. Columns of x must have the same names as columns of codes.

Details

The function returns the winner neuron in codes for each test vector in `x`. `codes` and `x` are one vector per row and must have the same number of columns (i.e. dimensions) and the identical column names.

`som.nn.visual` is the work horse for the k-NN-like classifier and normally used from `predict`.

Value

```
\code{data.frame} with 2 columns:
  \itemize{
    \item Index of the winner neuron for each row (index starting at 1).
    \item Distance between winner and row.
  }
```

SOMnn-class

An S4 class to hold a model for the topological classifier som.nn

Description

Objects of type SOMnn can be created by training a self-organising map with [som.nn.train](#).

Slots

`name` optional name of the model.

`date` time and date of creation.

`codes` `data.frame` with codebook vectors of the som.

`qerror` sum of the mapping errors of the training data.

`class.idx` column index of column with class labels in input data.

`classes` character vector with names of categories.

`class.counts` `data.frame` with class hits for each neuron.

`class.freqs` `data.frame` with class frequencies for each neuron (freqs sum up to 1).

`norm` logical; if TRUE, data is normalised before training and mapping. Parameters for normalisation of training data is stored in the model and applied before mapping of test data.

`norm.center` vector of centers for each column of training data.

`norm.scale` vector of scale factors for each column of training data.

`confusion` `data.frame` with confusion matrix for training data.

`measures` `data.frame` with classes as rows and the columns sensitivity, specificity and accuracy for each class.

`accuracy` The overall accuracy calculated based on the confusion matrix `cmat`: $acc = \text{sum}(\text{diag}(\text{cmat})) / \text{sum}(\text{cmat})$.

`xdim` number of neurons in x-direction of the som.

`ydim` number of neurons in y-direction of the som.

`len.total` total number of training steps, performed to create the model.

`toroidal` logical; if TRUE, the map is toroidal (i.e. borderless).

`dist.fun` function; kernel for the kNN classifier.

`max.dist` maximum distance for the kNN classifier.

`strict` Minimum vote for the winner (if the winner's vote is smaller than `strict`, "unknown" is reported as class label (default = 0.8)).

Index

`dist.fun.bubble`, [3](#), [22](#)
`dist.fun.inverse`, [3](#), [22](#)
`dist.fun.linear`, [4](#), [22](#)
`dist.fun.tricubic`, [5](#), [22](#)
`dist.torus`, [5](#)

`initialize`, ANY, ANY-method
 (`initialize`, SOMnn-method), [6](#)
`initialize`, SOMnn-method, [6](#)

`legend`, [10](#)

`norm.linear`, [8](#)
`norm.softmax`, [9](#)

`par`, [10](#)
`plot`, SOMnn, ANY-method, [9](#)
`plot`, SOMnn-method
 (`plot`, SOMnn, ANY-method), [9](#)
`predict`, SOMnn-method, [11](#)

`round.proBABILITIES`, [12](#)

SOM, [16](#), [19](#), [23](#)
`som`, [16](#), [19](#), [23](#)
`som.nn` (`som.nn`-package), [2](#)
`som.nn`-package, [2](#)
`som.nn.accuracy`, [13](#)
`som.nn.all.accuracy`, [14](#)
`som.nn.confusion`, [14](#)
`som.nn.continue`, [2](#), [15](#)
`som.nn.export.kohonen`, [17](#)
`som.nn.export.som`, [18](#)
`som.nn.multitrain`, [18](#)
`som.nn.set`, [21](#), [25](#)
`som.nn.train`, [2](#), [7](#), [22](#), [27](#)
`som.nn.validate`, [25](#)
`som.nn.visual`, [26](#)
`somgrid`, [16](#), [20](#), [24](#)
SOMnn (SOMnn-class), [27](#)
SOMnn-class, [27](#)