Package 'SOMDisco'

October 13, 2020

Type Package

Title Training and Visualizing a Conscience Self-Organizing Map

Version 1.0

Date 2020-04-25

Author Josh Taylor

Maintainer Josh Taylor <josh@somdisco.com>

Description SOMDisco makes the Conscience-SOM algorithm of DeSieno (1988) available to the machine learning community, leveraging Rcpp, RcppArmadillo and RcppParallel for speed. Additionally, extended visualizations of learned SOM products are provided to faciliate CSOM-based cluster extraction.

License GPL (>= 2)

Imports Rcpp (>= 1.0.1),
stringr,
igraph,
dplyr,
gtools,
Rdpack

LinkingTo Rcpp, RcppArmadillo, RcppParallel

RoxygenNote 7.1.1

Suggests knitr,
rmarkdown,
TopoRNet

VignetteBuilder knitr

RdMacros Rdpack

LazyData true

R topics documented:

as_list 3
build_CADJ 3
build_ctab 4
calc_Entropy 4
calc_eta 5
calc_SOM_fences 5
CONN 6

default_LRAS 7
distmat 7
find_BMU 8
find_RF_label 10
find_RF_members 10
geodesicdist 11
get_LRAS 11
initialize_SOM 12
linscale 12
load 13
load_list 14
map_from_netrng 14
map_to_netrng 15
new 15
recall_SOM 16
save 17
set_ctab 17
set_lattice 18
set_lattice_fences 18
set_LRAS 19
set_monitoring_freq 20
set_nBMU 21
set_netrng 21
set_p 22
set_parallel 22
set_p_equal 23
set_train_order 23
set_W 24
set_W_runif 25
set_X_label 25
SHGR 26
SOM 26
summarystat_RF_fwdmap 30
summarystat_RF_revmap 31
tableau20 32
tile_interior_point 32
train_SOM 33
update_learning_rates 34
update_p 34
update_W 35
vis_ctab 35
vis_LRAS 36
vis_som_annotate 36
vis_som_CADJvis 37
vis_som_CONNvis 38
vis_som_fences 39
vis_som_gradient 40
vis_som_label 42
vis_som_labeldist 43
vis_som_mUMatrix 43
vis_som_neurons 45
vis_som_prototypes 46

vis_som_setup 47
vis_som_tiles 47
vis_som_TopoView 48
vis_som_training 49

Index **51**

`as_list` *Convert an SOM object to a list*

Description

This method extracts all fields of an SOM object and places their stored values into the fields of an R list, with list field names matching the SOM object field names.

Usage

```
SOMobj$as_list()
```

Value

A list

`build_CADJ` *Build a CADJ matrix*

Description

Build a CADJ matrix

Usage

```
build_CADJ(BMU, nW, parallel = TRUE)
```

Arguments

`BMU`	matrix (nrow = nrow(X), ncol = nBMU) of indices (1-based) of the BMUs of each data vector in X.
`nW`	the number of SOM prototypes (CADJ matrices have nrow = ncol = nW)
`parallel`	optional, whether to compute in parallel. Default = TRUE

Details

The input BMU matrix should be the result of calling `find_BMU` with `nBMU` >= 2.

Value

a CADJ matrix

build_ctab — *Build a color table*

Description

Returns a default `ctab` given a set of (possibly non-unique) labels. The returned object is a data frame with columns `label` and `color` defining the color mapping.

Usage

```
build_ctab(labels)
```

Arguments

`labels`	a set of data labels to whose unique values colors will be assigned

calc_Entropy — *Calculate the Normalized Entropy of the SOM mapping*

Description

The normalized entropy of the SOM quantization is given by

$$entropy = -sum(F * log(F))/log(nW)$$

where `F` is a vector of Receptive Field relative frequencies (i.e., RF_size/nX).

SOMs trained with the CSOM update rule seek to maximize the entropy of the learned mapping, so this normalized entropy measure provides a way of comparing different SOMs regardless of their size (or the number of training vectors available).

It is usually not necessary to directly call this function as it is invoked inside recall_SOM.

Usage

```
SOMobj$calc_Entropy()
```

Value

None, the field `Entropy` is set internally.

calc_eta — *Calculate the SOM neighborhood function*

Description

Topology preservation in SOM mappings is enforced by the neighborhood function, which, during each training step, specifies cooperative updates of a small radius of prototype's whose neurons neighbor the BMU. The maximum neighborhood radius at each time is set by the `sigma` parameter specified in set_LRAS. The eta coefficients are computed from a logistic decay up to this `sigma`, so that during each training step the winning prototype is updated with the strongest effect, the prototypes within a radius = 1 on the lattice are updated with the next strongest effect, and so on (up to a maximum radius of `sigma`).

It is usually not necessary to directly call this function as it is invoked inside train_SOM.

Usage

```
SOMobj$calc_eta(sigma)
```

Arguments

`sigma`	the maximum neighborhood radius for which the neighborhood functional is applied

Value

a vector of the eta neighborhood coefficients for a given value of `sigma`. The vector is ordered such that `eta[1]` is the coefficient applied to the prototype update of the BMU, `eta[2]` is the coefficient applied to prototype updates within a radius=1 of the BMU, `eta[3]` is the coefficient applied to prototype updates within a radius=2 of the BMU, etc.

calc_SOM_fences — *Calculate the fences of the SOM lattice*

Description

Lattice fences between neighboring neurons on the lattice are the (squared) Euclidean distance between the neurons' corresponding prototypes.

Usage

```
calc_SOM_fences(nu_xy, W, nu_verts, nu_ADJ, parallel = TRUE)
```

Arguments

`nu_xy`	a matrix whose rows contain the (x,y) coordinates (cols 1 and 2, respectively) of each lattice neuron, ordered from the bottom-left of the lattice.
`W`	the prototype matrix (rows ordered the same as nu_xy)
`nu_verts`	a 3-d array whose slices contain the vertices of each lattice tile (slices ordered the same as nu_xy)
`nu_ADJ`	the neuron adjacency matrix
`parallel`	optional, whether to compute in parallel. Default = TRUE

Value

A data frame identifying the fence and its corresponding value between each pair of neighboring lattice neurons, with columns:

- i the index of the 1st neuron comprising the fence
- j the index of the 2nd neuron comprising the fence
- x0 the lattice x-coord of neuron i
- y0 the lattice y-coord of neuron i
- x1 the lattice x-coord of neuron j
- y1 the lattice y-coord of neuron j
- value the fence value separating neurons i and j

`CONN` *The CONNectivity matrix of SOM prototypes*

Description

The CONN matrix is calculated from the `CADJ` matrix computed during recall_SOM:

$$CONN = CADJ + t(CADJ)$$

.

`CADJ` describes the strength of the topological connectivites between SOM prototypes in input space (higher CADJ values between prototypes w_i and w_j indicate stronger evidence that `w_i` and `w_j` belong to the same cluster). The (i,j) values of CADJ are defined:

$$CADJ_{ij} = \#(BMU1(X) = iandBMU2(X) = j)$$

By this definition, `CADJ` is an asymmetric adjacency matrix. CONN is its symmetric counterpart.

Usage

```
SOMobj$CONN()
```

Value

the CONN matrix (nrow = ncol = `nW`)

References

Taşdemir K, Merényi E (2009). "Exploiting Data Topology in Visualization and Clustering of Self-Organizing Maps." *IEEE Transactions on Neural Networks*, **20**(4), 549-562. ISSN 1045-9227, doi: 10.1109/TNN.2008.2005409.

`default_LRAS` *Generate a default Learning Rate Annealing Schedule*

Description

Generate a default Learning Rate Annealing Schedule

Usage

```
default_LRAS(nX)
```

Arguments

`nX`	the number of data vectors used for training

Value

the Learning Rate Annealing Schedule as a data frame with columns t (time step), alpha (prototype update rate), beta (win frequencies update rate), gamma (bias update rate), sigma (geodesic lattice distance to which prototype update is applied)

`distmat` *Compute distances between rows of two matrices*

Description

Compute distances between rows of two matrices

Usage

```
distmat(
  X1,
  X2,
  which_dist = "L22",
  parallel = TRUE,
  bias = NULL,
  X1min = NULL,
  X1max = NULL,
  X2min = NULL,
  X2max = NULL
)
```

Arguments

`X1`	a data matrix, vectors in rows
`X2`	a data matrix, vectors in rows. ncol(X2) must equal ncol(X1).
`which_dist`	optional, the distance to compute. Options are

- "L2" for Euclidean,
- "L22" for Squared Euclidean",

- "L1" for L-1 distance,
- "LInf" for L-Inf distance

Default = "L22".

parallel
: optional, whether to compute in parallel. Default = TRUE

bias
: optional, a vector of bias to be applied to the distance calculation when computing BMU. Default = NULL, meaning no bias is applied. If given, length(bias) must equal nrow(X2) (one bias for every row of X2).

X1min
: optional, a vector giving the minimum of the range of X1 used in any linearly scaling before distances are computed. Default = NULL, meaning no scaling is applied.

X1max
: optional, a vector giving the maximum of the range of X1 used in any linearly scaling before distances are computed. Default = NULL, meaning no scaling is applied.

X2min
: optional, a vector giving the minimum of the range of X2 used in any linearly scaling before distances are computed. Default = NULL, meaning no scaling is applied.

X2max
: optional, a vector giving the maximum of the range of X2 used in any linearly scaling before distances are computed. Default = NULL, meaning no scaling is applied.

Details

If given, bias affects the distance calculation between the rows of X1 and X2 by: which_dist(X1[i,], X2[j,]) - bias[j].

The inputs X1min, X1max, X2min, X2max control any linearly scaling (from the range of X1, to that of X2) that is applied prior to distance calculation. If any of the above are given, they all must be given. Each can be given as either a vector controlling ranges by dimension (length = ncol(X1)), or a length 1 vector (in which case the single value will be recycled across dimensions).

Specifying scaling forces distance computation to be performed in the X2 range. This is an optional feature.

Value

a matrix whose (i,j) entry contains the requested distance between the i-th row of X1 and and j-th row of X2

`find_BMU` *Find BMU of data vectors*

Description

Find BMU of data vectors

Usage

```
find_BMU(
  X,
  W,
  nBMU = 2L,
  parallel = TRUE,
  bias = NULL,
  Xmin = NULL,
  Xmax = NULL,
  Wmin = NULL,
  Wmax = NULL
)
```

Arguments

`X`	a data matrix, vectors in rows
`W`	a prototype matrix, vectors in rows
`nBMU`	optional, the number of BMUs returned. Default = 2.
`parallel`	optional, whether to compute in parallel. Default = TRUE
`bias`	optional, a vector of bias to be applied to the distance calculation when computing BMU. Default = NULL, meaning no bias is applied. If given, length(bias) must equal nrow(W) (one bias for every prototype).
`Xmin`	optional, a vector giving the minimum of the range of X used in any linearly scaling before distances are computed. Default = NULL, meaning no scaling is applied.
`Xmax`	optional, a vector giving the maximum of the range of X used in any linearly scaling before distances are computed. Default = NULL, meaning no scaling is applied.
`Wmin`	optional, a vector giving the minimum of the range of W used in any linearly scaling before distances are computed. Default = NULL, meaning no scaling is applied.
`Wmax`	optional, a vector giving the maximum of the range of W used in any linearly scaling before distances are computed. Default = NULL, meaning no scaling is applied.

Details

If given, bias affects the distance calculation between the rows of X and W by: dist(X[i,], W[j,]) - bias[j].

The inputs Xmin, Xmax, Wmin, Wmax control any linearly scaling that is applied prior to distance calculation. If any of the above are given, they all must be given. Each can be given as either a vector controlling ranges by dimension (length = ncol(W)), or a length 1 vector (in which case the single value will be recycled across dimensions). Specifying the X and W ranges is only useful if you are trying to match a BMU selection that was performed during SOM training, which utilized an external / internal network range.

Value

a list with components

- BMU a matrix (nrow = nrow(X), ncol = nBMU) whose (i,j) entry lists the index (the row of W, 1-based) of the j-th BMU of the i-th row of X.
- SQE a vector (length = nrow(X)) whose i-th element gives the Squared Quantization Error of the i-th row of X

`find_RF_label` *Compute the plurality label of each prototype.*

Description

For labeled data mapped to each prototype's Receptive Field, the plurality label is returned.

Usage

```
find_RF_label(X_label, RF_members, parallel = TRUE)
```

Arguments

`X_label`	a vector (length = nrow(X)) of data labels
`RF_members`	a list containing the (1-based) indices of the data vectors mapped to each prototype, as returned by `find_RF_members`.
`parallel`	optional, whether to compute in parallel. Default = TRUE

Value

a vector of prototype labels. If a prototype's Receptive Field is empty the returned label is NA

`find_RF_members` *Produce a list of data vectors in each prototype's Receptive Field*

Description

Produce a list of data vectors in each prototype's Receptive Field

Usage

```
find_RF_members(BMU, nW, parallel = TRUE)
```

Arguments

`BMU`	matrix (nrow = nrow(X), ncol = nBMU) of indices (1-based) of the BMUs of each data vector in X.
`nW`	the number of SOM prototypes (CADJ matrices have nrow = ncol = nW)
`parallel`	optional, whether to compute in parallel. Default = TRUE

Details

The input BMU matrix should be the result of calling `find_BMU` with `nBMU` >= 2.

Value

a list (length = nW), containing the (1-based) indices of the data vectors mapped to each prototype.

`geodesicdist` *Shortest path distances between vertices of a graph*

Description

This is a wrapper for igraph::distances, see its help for more information.

Usage

```
geodesicdist(ADJ, weighted = FALSE, directed = FALSE)
```

Arguments

`ADJ`	the adjacency matrix of graph vertices. Can be weighted.
`weighted`	optional, whether to consider the edge weights found in ADJ and construct a weighted graph. Default = FALSE.
`directed`	optional, whether to consider the graph as directed. Default = FALSE.

Value

a shortest-path distance matrix between graph vertices (nrow = ncol = num. vertices)

`get_LRAS` *Get the Learning Rate Annealing Schedule*

Description

A getter function for the LRAS (described in set_LRAS) stored in the SOM object.

Usage

```
SOMobj$get_LRAS()
```

Value

the effective LRAS of the SOM, as a data frame.

initialize_SOM *Initialize a SOM object*

Description

Sets up a SOM lattice for training, populates default training parameters, and attaches the training data to the specified SOM.

Usage

```
SOMobj$initialize_SOM(X, som_x, som_y, lattice_type)
```

Arguments

X	matrix of training data (data vectors in rows)
som_x	width of the SOM (number of neurons)
som_y	height of the SOM (number of neurons)
lattice_type	either "grid" for rectangular lattices or "hex" for hexagonal lattices

Details

This is a wrapper function to perform all steps necessary to setup a SOM for learning. Internally it calls set_lattice, set_netrng, set_W_runif, set_p_equal, and set_LRAS, all with their default arguments. It also sets internal variables:

- `som_x`, `som_y`, `nW`, `lattice_type`
- `d`, `nX`, `X_stats`
- `nBMU`

Value

None, data and lattice related fields are set internally

linscale *Linear scaling of the rows of a data matrix*

Description

Each row is mapped linearly from a given range, to a given range

Usage

```
linscale(X, from_min, from_max, to_min, to_max, parallel = TRUE)
```

Arguments

`X`	a data matrix with vectors in rows
`from_min`	a vector defining the min from which X is scaled, by dimension
`from_max`	a vector defining the max from which X is scaled, by dimension
`to_min`	a vector defining the min to which X is scaled, by dimension
`to_max`	a vector defining the max to which X is scaled, by dimension
`parallel`	boolean, whether to compute in parallel

Value

a matrix (same dimensions as X) whose rows are scaled to the requested range

---

`load` *Load an existing SOM object*

Description

SOM objects previously written to disk via the save method can be re-loaded into a new R environment with this function. All fields of the internal C++ class will be populated, and all methods can be called on the loaded SOM object.

Usage

```
SOMobj$load(somfile)
```

Arguments

`somfile`	a string indicating the file path and name of the saved SOM object.

Details

Because the .som file is in .rds format it can, technically, be loaded directly into an R environment as a list via readRDS. This can be useful for spot checking the contents of a saved SOM object, but does not allow use of any of its methods (or visualizations). The `load` methods allows for proper restoration of a previously saved SOM.

Value

None, the SOM object is loaded

load_list *Populate a SOM object from a list*

Description

This method populates all fields of a SOM object from the fields of an R list object. The list must have field names which exactly match the SOM field names.

Usage

```
SOMobj$load_list(SOMList)
```

Arguments

SOMList	a SOM object converted to a list, e.g., with `as_list`

Value

None

map_from_netrng *Map a prototype matrix to external network range*

Description

Provides reverse functionality of map_to_netrng, to linearly scale vectors from internal to external network range.

Usage

```
SOMobj$map_from_netrng(W)
```

Arguments

W	a matrix occupying internal network range to be mapped to external network range. `W` can be either a single vector, or a matrix with multiple vectors in its rows. It must have the same dimension (number of columns) as the SOM (which is stored internally as `d`). Note that the SOM prototypes are computed and stored in internal network range; to compare them directly to the training data they should be mapped to the external range via this function.

Value

a matrix (same dimension as input) containing the scaled data vectors in its rows `map_to_netrng` *Map a data matrix to internal network range*

Description

During training, prototype updates and BMU selelction are performed in the internal network range. This function allows mapping an arbitrary data matrix to this range to, e.g., replicate BMU selection or be able to compare data to prototypes.

Usage

```
SOMobj$map_to_netrng(X)
```

Arguments

`X`	a data matrix (occupying the external network range) to be mapped to the internal network range. `X` can be either a single vector, or a matrix with multiple data vectors in its rows. It must have the same dimension (number of columns) as the SOM (which is stored internally as `d`).

Value

a matrix (same dimension as input) containing the scaled data vectors in its rows

`new` *Create an empty SOM object*

Description

Create an empty SOM object

Usage

```
SOM$new()
```

Value

an empty SOM object templated for initialization (via initialize_SOM) and training (via train_SOM).

recall_SOM *Recall a trained SOM object*

Description

A SOM recall maps all training data to their representative prototypes (their BMUs). Several quantities, as outlined below, result from this mapping.

Usage

```
SOMobj$recall_SOM(X)
```

Arguments

X	the training data matrix. This should be the same matrix that was input to initialize_SOM (checks will be performed on its statistics, if they do not match an error will be returned).

Details

Several internal fields are set by the recall function:

- `BMU` matrix (nrow = `nX`, ncol = `nBMU`) containing the 1st, 2nd, ... BMUs for every training vector in its columns. The valued in `BMU` are the 1-based indices of the prototypes, in neuron order.
- `SQE` vector (length = `nX`) containing the squared quantization error of each training vector, as quantized by its 1st BMU
- `CADJ` the CADJ matrix (nrow = ncol = `nW`) of Cumulative (weighted) Topological Adjacencies of the SOM prototypes. See CONN for details.
- `RF_size` vector (length = `nW`) containing the number of training vectors mapped to each prototype
- `Entropy` the (normalized) entropy of the discrete SOM mapping
- `RF_members` a list (length = `nW`) of the training vector indices mapped to each prototype
- `fences` a data frame containing the information for visualizing the U-matrix fences on the SOM lattice, see set_lattice_fences for details.
- `RF_label` and `RF_label_dist`, the values of `X_label` propagated to the Receptive Fields, according to the SOM mapping.

Note: the "Receptive Field" of a prototype w_j is the set of all training data for which w_j is the BMU.

It is usually not necessary to directly call this function as it is invoked inside train_SOM.

Value

None, the fields described above are calculated and stored save *Save a SOM object*

Description

All fields in a SOM object can be saved to disk with this function, which allows them to be re-loaded into a new R environment at a later time (for analysis, or possibly extended training).

Usage

```
SOMobj$save(somfile)
```

Arguments

`somfile`	a string indicating the file path and name in which to save the SOM object. This must end in extension ".som", otherwise an error is returned.

Details

The SOM object is saved to disk as an R list, with each field occupying a corresponding field of the list. The file is saved in .rds format (can check its details with infoRDS).

Saved SOMs can be re-loaded with load

Value

None, the SOM object is saved to disk

set_ctab *Set the color table*

Description

The color table controls the mapping of the unique labels found in `X_label` to distinct colors.

Usage

```
SOMobj$set_ctab(ctab)
```

Arguments

`ctab`	a color table defining the mapping between the unique labels found in `X_label` and distinct colors. Must be a data frame with columns 'label' and 'color' (where 'color' is in HEX format).

Details

`ctab` must contain color mappings for all unique labels found in `X_label`. A check will be performed internally.

Value

None, field `ctab` is set internally

`set_lattice` *Setter function for the SOM lattice.*

Description

Sets the neuron lattice coordinates and associated quantities needed for SOM training and visualization.

Usage

```
SOMobj$set_lattice()
```

Details

initialize_SOM must be called prior to calling `set_lattice`.

Internally, the following fields are computed and set:

- `nu_xy` the (x,y) coordinates of neurons on the lattice
- `nu_ij` the (i=row,j=col) coordinates of neurons on the lattice
- `nW` the number of neurons / prototypes
- `nu_ADJ` the adjacency matrix of neurons on the lattice (identifying immediate neighbors)
- `nu_nhblist` an organized hierarchical list of neurons sorted by their lattice distance from each other. `nu_nhblist[[i]][[j]]` contains a vector of neuron IDs that are distance j from neuron i on the lattice.
- `nu_verts` cube whose slices contain the vertices of each lattice tile. Slice i contains the vertices of lattice tile i in its rows, in vertex order.

Value

None, lattice related fields are set internally

`set_lattice_fences` *Compute and store the U-Matrix fences*

Description

The U-Matrix fences, when visualized on the SOM lattice, are an easy way to determine the similarity between neighboring lattice-neighboring prototypes. This function computes a data frame containing the information required to visiualize such fences, which can be accomplished via vis_som_fences.

It is usually not necessary to directly call this function as it is invoked inside recall_SOM.

Usage

```
SOMobj$set_lattice_fences()
```

Value

None, data frame `fences` is set internally

References

Ultsch A, Siemon HP (1990). "Kohonen's Self Organizing Feature Maps for Exploratory Data Analysis." In Widrow B, Angeniol B (eds.), *Proceedings of the International Neural Network Conference (INNC-90), Paris, France, July 9–13, 1990 1. Dordrecht, Netherlands*, volume 1, 305–308. http://www.uni-marburg.de/fb12/datenbionik/pdf/pubs/1990/UltschSiemon90.

`set_LRAS` *Set the Learning Rate Annealing Schedule*

Description

The LRAS is a data frame with columns:

- t
- alpha
- beta
- gamma
- sigma

The rows of the data frame specify the learning parameters `alpha`, `beta`, `gamma` and `sigma` that are in effect up to (but not including) the training step identified in `t`. At each training step alpha controls the strength of the prototype update, beta controls the strength of the win frequency update, gamma controls the magnitude of the entropy-maximizing bias, and sigma controls the maximum radius of the neighborhood function eta.

Internally, the columns of the input are extracted and stored in a `std::map` for quick lookup. At each training step, effective values from this map are extracted and stored in fields `alpha`, `beta`, `gamma` and `sigma` (the latter, in turn, also updates the neighborhood function `eta`).

A default LRAS, based on the number of training sampled, can be obtained via default_LRAS.

Internally, a final time step `t = std::max_element<unsigned int>` is appended to the LRAS, which has the effect of recycling the last set of learning parameters for indefinite training.

Kohonen's original SOM algorithm can be achieved by setting both beta and gamma = 0 in the LRAS.

Usage

```
SOMobj$set_LRAS(LRAS)
```

Arguments

`LRAS`	a data frame, which must have the above structure.

Value

None, the LRAS is stored internally

References

DeSieno D (1988). "Adding a conscience to competitive learning." In *IEEE international conference on neural networks*, volume 1(6), 117–124. IEEE Piscataway, NJ.

`set_monitoring_freq` *(De-)Active Monitoring of SOM Training*

Description

Various quantities can be computed and stored at regular training step intervals to allow the analyst to observe how SOM learning progresses. This function either activates or de-activates this monitoring.

Usage

```
SOMobj$set_monitoring_freq(mtr_freq)
```

Arguments

`mtr_freq`	the incremental training step count at which monitoring snapshots are taken and stored during SOM training. If = 0, which is the default set when calling initialize_SOM, then no monitoring is performed.

Details

As monitoring proceeds, the internal field `mtr_age` is updated with the time (training) steps at which snapshots were taken.

If activated, monitoring computes and stores the following quantities:

- `mtr_RMSQE` a matrix (nrow = length(`mtr_age`), ncol = `nW+1`) containing the Root Mean Squared Error of the quantization of the data by its BMU prototype (the Root Mean Quantization Error at the prototype level). The last (`nW` + 1) column contains the RMSQE of the quantization over all prototypes.
- `mtr_QSI` a vector (length = length(`mtr_age`)) of the Quantization Stability Index computed during each monitoring snapshot. The QSI at any monitoring step is the proportion of data samples that have switched BMUs from the previous monitoring snapshot (by convention, QSI at the initial monitoring step = NA). QSI can convey when the SOM "settles down", and reaches a stable quantization.
- `mtr_Entropy` a vector (length = length(`mtr_age`)) of the normalized Entropies of the SOM mappings taken at each monitoring step.

The monitored quantities can be visualized after a training round by calling vis_som_training.

Value

None, field `mtr_freq` is set internally.

set_nBMU *Set the number of BMUs*

Description

The Best Matching Unit of a sample vector x is the prototype w_j (and its associated neuron) that minimizes Euclidean distance $d(x, w_j)$. The SOM requires calculating at least the first BMU during training and recall. Calculation of the CADJ matrix requires calculating and storing the second BMU as well (the second Best Matching Unit). This function sets the internal field nBMU, which specifies the number of BMUs stored in the field BMU during recall. Any number > 2 could be useful for further analysis. Be default, nBMU is set = 2 during initialize_SOM.

Usage

```
SOMobj$set_nBMU(nBMU)
```

Arguments

nBMU	the number of BMUs requested to compute

Value

None, nBMU is set internally.

set_netrng *Set the external/internal network range*

Description

Training (i.e., prototype updating) is performed in an "internal" network range. When presented to the network for training, each data vector is scaled linearly from the external data range to the internal data range. This function sets the min/max of both ranges.

Usage

```
SOMobj$set_netrng(ext_min, ext_max, int_min, int_max).
```

Arguments

ext_min	min of the external data range (the training data min)
ext_max	max of the external data range (the training data max)
int_min	min of the internal data range (the prototype min)
int_max	max of the internal data range (the prototype max)

Details

Both `ext_min` and `ext_max` are stored internally (as `netrng_ext_min` and `netrng_ext_max`, respectively) vectors of length `d` (data dimension). Thus, either can be input as either a length `d` vector (to specify a different range of linearly scaling across dimension), or as a single number (which will be recycled across dimension to produce a length `d` vector). `int_min` and `int_max` (which are stored internally as `netrng_int_min` and `netrng_int_max`, respectively) should be given as a single number (not a vector, varying internal ranges across dimension is not currently supported).

`set_netrng` is called by default during initialize_SOM using the observed min/max of the data as the external range, and 0/1 as the internal range.

Scaling can be turned off by setting `ext_min = int_min = 0` and `ext_max = int_max = 1`.

Value

None, scaling ranges are set internally

`set_p` *User-specified initialization of CSOM win frequencies*

Description

This function allows setting of the internal field `p`, which stores the CSOM win frequencies for each prototype. It is made available mostly for experimentation.

Usage

```
SOMobj$set_p(p)
```

Arguments

`p`	user-specified win frequency vector. Must have length = `nW`

Value

None, win frequency vector `p` is stored internally

`set_parallel` *Setter function for parallel computation.*

Description

Sets an internal flag controlling whether computations are performed in parallel.

Usage

```
SOMobj$set_parallel(parallel)
```

Arguments

`parallel`	either TRUE or FALSE, as desired

Details

Parallel computation is supported via RcppParallel, see setThreadOptions for details to control threading. By default, `parallel` = `TRUE` at SOMobj instantiation.

Value

None, `parallel` field is set internally

`set_p_equal` *Equi-probable initialization of CSOM win frequencies*

Description

The Conscience-SOM algorithm of DeSieno attempts a Maximum-Entropy SOM mapping by updating a record of each prototype's win frequency. This roughly mimics the proportion of times during training that each prototype is selected as some datum's BMU. This function initializes the win frequencies for all prototypes to 1 / `nW`, which are stored internally in field `p`.

Note: this function is called automatically by initialize_SOM.

Usage

```
SOMobj$set_p_equal()
```

Value

None, win frequency vector `p` is stored interally.

References

DeSieno D (1988). "Adding a conscience to competitive learning." In *IEEE international conference on neural networks*, volume 1(6), 117–124. IEEE Piscataway, NJ.

`set_train_order` *Set the sample training order*

Description

Usually during training, sample vectors should be drawn randomly (which train_SOM does by default, and stores a record of the drawing order here). If a specific training order is desired (for, e.g., experimental purposes), this can be set by calling `set_train_order` prior to calling `train_SOM`. This forces data presentation during SOM training to follow the prescribed ordering. If a specific training order is set, it is only valid during the subsequent call to `train_SOM`; if multiple rounds of training are performed, each with a desired training order, then `set_train_order` must be called again prior to each call to `train_SOM`.

Usage

```
SOMobj$set_train_order(train_order)
```

Arguments

train_order	a vector containing the (1-based) indices (i.e., row indices of training data X) which will govern the presentation order during training.

Details

If set_train_order is called it will override the specification of the nsteps argument during a subsequent call to train_SOM. That is, the entire set of training data specified by input train_order will be used during SOM training, regardless of the number of training steps requested during the call to train_SOM.

The internal logical field user_train_order tracks whether this function has been called.

Value

None, train_order and user_train_order are set internally.

---

set_W *Setter function for the prototype matrix* W

Description

Random prototype initialization is performed via set_W_runif. This function allows setting W to user-prescribed values. If the input values reside outside of the internal network range a warning will be issued

Usage

```
SOMobj$set_W(W)
```

Arguments

W	matrix of prototype values, with rows following neuron ordering. nrow(W) must equal nW and ncol(W) must equal d, otherwise an error will be returned.

Value

None, prototype matrix W is set internally.

`set_W_runif` *Random uniform initialization of SOM prototypes*

Description

SOM training typically begins with randomly initialized prototype vectors. This function performs this random initialization in the range `mid` +/- 5 where `mid` is the midpoint of `netrng_int_min` and `netrng_int_max`.

To force random sampling according to a specified seed, call R's set.seed function just prior to calling `set_W_runif`

Note: this function is called automatically by initialize_SOM.

Usage

```
SOMobj$set_W_runif()
```

Value

None, prototype matrix `W` is initialized internally.

`set_X_label` *Set the training data labels*

Description

If the training data possess labels, this method stores them in the `X_label` field, which is used to propagate the `RF_label` and `RF_label_dist` fields during `recall_SOM`.

Usage

```
SOMobj$set_X_label(X_label, ctab)
```

Arguments

`X_label`	a vector (length = `nW`) containing character labels for each training observation.
`ctab`	a color table defining the mapping between the unique labels found in `X_label` and distinct colors. Must be a data frame with columns 'label' and 'color' (where 'color' is in HEX format).

Details

During `initialize_SOM` the data labels are set to a default value of "?". This method allows changing these defaults. The function `build_ctab` can be used to construct a default color table, if needed.

Value

None, field `X_label` is set internally.

SHGR *A SHGRWalk Synthetic Hyperspectral Image Cube*

Description

The SHGRWalk (SHGR, for short) data suite is a collection of Synthetic Hyperspectral 128 x 128 pixel image cubes whose individual pixel "spectra" were sampled from a multi-component Gaussian Mixture Model whose component means were set via a (secondary) Gaussian Random Walk across the synthetic "spectral channels" (the data dimension, `d`). Individual component variances possesses a Toeplitz correlation structure with various noise levels. After sampling the pixels were labeled according to which mixture component (class) they were sampled from, and then organized into the 128 x 128 pixel image such that pixels from the same sampling component occupy contiguous blocks within the overall image. More information about the SHGRWalk data can be found here `need link`.

Usage

```
SHGR
```

Format

For demonstration of SOM learning a 100-dimensional SHGR cube with 20 distinct classes (each with correlated noise) has been included in the `SOMDisco` package. These data and their associated metadata are stored in a list variabled named `SHGR` with components:

X data matrix whose 16,384 rows represent the 128 x 128 pixels in the image, and 100 columns represent the 100 spectral channels at which the synthetic reflectances were measured

label character vector (length = 16,384) containing class labels of each row of `X`, denoted by the letters `A` -`T`

ctab color table mapping the unique character labels in the dataset to a pre-defined representative color. The color table is stored as a data frame with columns `label` and `color`.

pxl.coords matrix (nrows = 16,384) whose two columns give the `(row,col)` pixel indices of the rows of `X` in the image. This information is needed to map the data matrix `X` back to its data cube format.

identifier string detailing the specifics of the example SHGR cube

SOM *The SOM Object Class*

Description

The SOM Object Class

Fields

`age` The current age of the SOM (the number of training steps that have been performed thus far).

`alpha` The strength of the prototype update, as set in the LRAS, given the current age of the SOM.

`beta` The strength of the win frequency update, as set in the LRAS, given the current age of the SOM.

`bias` Vector of CSOM biases for each prototype.

`BMU` Matrix of BMUs (the best matching prototype indices) for each training data vector. nrow = nX, ncol = nBMU.

`CADJ` The Cumulative Adjacency Matrix of SOM prototypes.

`ctab` The color table which maps unique labels found in `X_label` to colors. Must be a data frame with columns 'label' and 'color'.

`d` The training data (and prototype) dimension

`Entropy` The normalized entropy of the learned SOM quantization

`eta` Vector of coefficients applied to cooperate lattice neighbor updates during training. The elements of eta exhibit logistic decay as lattice distance between neurons increases (and cease to effect any prototype updates for neurons beyond distance `sigma` from a BMU).

`fences` A dataframe storing the information required to visualize the U-Matrix fences on the SOM lattice.

`gamma` The strength of the win frequencies on the prototype biases, as set in the LRAS, given the current age of the SOM.

`is_netrng_set` Flag indicating whether `set_netrng` has been called.

`is_protos_init` Flag indicating whether `set_W_runif` or `set_W` has been called.

`is_recalled` Flag indicating whether `recall_SOM` has been called since last call to `train_SOM`.

`is_trained` Flag indicating whether `train_SOM` has been called since prototypes have been initialized.

`is_winfrq_init` Flag indicating whether `set_p_equal` or `set_p` has been called.

`lattice_type` String defining the SOM lattice topology, either "hex" or "grid".

`LRAS` The Learning Rate Annealing Schedule, as a data frame with columns `t`, `alpha`, `beta`, `gamma`, `sigma`. See default_LRAS for an example.

`mtr_age` A vector of SOM ages at which monitoring snapshots were taken during training.

`mtr_Entropy` A vector of SOM (normailized) entropies at the monitoring snapshots.

`mtr_freq` The incremental number of steps at which monitoring snapshots are taken during training. Set = 0 to disable monitoring.

`mtr_QSI` A vector of the Quantization Stability Indices computed at each monitoring snapshot. QSI is the proportion of data vectors whose BMUs have changed since the last monitoring snapshot was taken.

`mtr_RMSQE` A matrix of Root Mean Square Quantization Errors at the prototype level (columns) at each monitoring snapshot (rows). This matrix has `nW+1` columns, were the last contains the global RMSQE (average over all prototype-level RMSQEs).

`nBMU` The number of BMUs recorded during `recall_SOM`. The first BMU for a training data vector is its closest SOM prototype, the 2nd BMU is the next-closest prototype, etc.

`netrng_ext_max` The max of the training data range, used for network scaling.

`netrng_ext_min` The min of the training data range, used for network scaling.

`netrng_ext_rng` The entire range (max - min) of the external data range.

netrng_int_max The max of the internal (network) range, used for network scaling.

netrng_int_min The min of the internal (network) range, used for network scaling.

netrng_int_rng The entire range (max - min) of the internal range.

nu_ADJ Adjacency matrix of lattice neurons, as dictated by the lattice_type.

nu_ij Matrix of (i=row,j=col) coordinates of neurons on the SOM lattice.

nu_nhblist A list of list describing shortest-path neuron distances according to the SOM lattice topology set in lattice_type. nu_nhblist[[i]][[j]] contains all neuron indices that are lattice distance j from neuron i.

nu_verts Cube whose slices contain the (x,y) coordinates of each lattice tile.

nu_xy Matrix of (x,y) coordinates of neurons on the SOM lattice, primarily used for visualizations.

nW Number of SOM prototypes (and neurons), = som_x * som_y.

nX Number of training data vectors (nrows of the training data matrix X).

p Vector of the win frequencies of each prototype, as defined in the Conscience-SOM algorithm.

parallel Flag indicating whether computations should be performed in parallel, using the RcppParallel package.

RF_label Character vector of prototype labels, derived from the labels of training data mapped to each prototype. Only valid if the training data possesses labels.

RF_members A list of training data indices mapped to each prototype during recall_SOM.

RF_size A vector storing the sizes of each prototype's Receptive Field (the number of training vectors mapped to each prototype during recall_SOM).

sigma The maximum lattice distance between a BMU and its lattice neighborhood to which cooperative SOM prototype updates are applied.

som_x Width of SOM lattice, in neurons

som_y Height of SOM lattice, in neurons

SQE Vector of the Squared Quantization Error resulting from quantizing each training vector by its BMU (length = nW).

train_order Vector of indices of training vectors recording the order in which data were drawn to be presented to the network during training.

user_train_order Flag indicating whether a user-specified training order has been set prior to training.

W The prototype matrix (nrow = nW, ncol = d) storing SOM prototypes in its rows. The rows are in neuron order, so row1 corresponds to the (i=1,j=1) lattice neuron, row2 corresponds to the (i=1,j=2) lattice neuron, and so on.

X_label A vector (length = nX) containing character labels of each training observation.

X_stats A vector storing simple statistics (min, max, value of 1st element, value of last element) of the entire training data X that was specified during initialize_SOM. Subsequent methods which require the training data matrix (both train_SOM and recall_SOM) check that their inputs match those computed and stored in X_stats to ensure the same training data is given to all SOM functions.

vis_par Stores the par() settings for base R graphics required to layer the vis_som_* functions atop each other. Usually this field is not needed by the user.

vis_tile_bg Stores the current coloring of the lattice tiles, which sets the text annotation color if requested as "auto". Usually this field is not needed by the user.

vis_xlim Stores the xlim of the SOM plotting window Usually this field is not needed by the user.

vis_ylim Stores the ylim of the SOM plotting window. Usually this field is not needed by the user.

Methods

Each class method has its own documentation, accessible via ?SOMDisco::<method_name>. For completeness, the list is repeated here in entirety. Additional functionality for visualizing a trained SOM object is available through the vis_som_* functions. See their documentation for more information.

calc_eta Calculate the coefficients governing cooperative updates of the lattice neighbors of a BMU.

CONN The symmetric Cumulative Adjacency Matrix of SOM prototypes.

calc_Entropy Compute the normalized entropy of the SOM quantization.

get_LRAS Get the Learning Rate Annealing Schedule.

initialize_SOM Setup a SOM object for training.

new Instantiate a new SOM object.

map_from_netrng Scale the rows of a matrix from the internal network range, to the external network range.

map_to_netrng Scale the rows of a matrix from the external network range, to the internal network range.

recall_SOM Recall a trained SOM object.

set_ctab Set the color table mapping unique data labels to colors.

set_lattice Set the SOM lattice quantities.

set_lattice_fences Compute and store the information needed to visualize U-Matrix fences on the SOM lattice.

set_LRAS Set the Learning Rate Annealing Schedule.

set_monitoring_freq (De-)Activate monitoring of SOM training.

set_nBMU Set the number of BMUs recorded during recall_SOM.

set_netrng Set the ranges used for linear network scaling.

set_p Set the CSOM win frequencies to user-specified values.

set_p_equal Set the CSOM win frequencies to the equiprobable value 1/nW.

set_parallel Set the parallel computation flag.

set_RF_label Set the prototype label based on labeled training data.

set_train_order Set the order in which training data are presented to the network in train_SOM.

set_W Set the SOM prototypes to user-specified values.

set_W_runif Initialize the SOM prototypes to uniform random values.

set_X_label Set the training data labels.

tile_interior_point Get a list of interior points within each SOM lattice tile. Useful mostly for visualization routines.

train_SOM Train the SOM object.

update_learning_rates Update the learning rates based on the current SOM age.

update_p Update the CSOM win frequencies during training.

update_W Update the prototypes during training.

save Save a trained SOM object to disk.

load Load a saved SOM object from disk into a current R environment.

as_list Convert and return all fields of a SOM object to an R list.

load_list Populate an instance of a SOM object from an R list.

summarystat_RF_fwdmap *Receptive Field Summary Statistics*

Description

Compute summary statistics of each Receptive Field of a vector quantizer, given its forward mapping. A forward mapping is a vector containing the prototype index to which each data vector is mapped.

Usage

```
summarystat_RF_fwdmap(X_vals, fwdmap, nW, stat = "mean", parallel = TRUE)
```

Arguments

`X_vals`	the (possibly multivariate) data values to be summarized, as a matrix (nrows = nrow(X), ncol = num. dimensions)
`fwdmap`	a vector (length = nrow(X)) of indices (1-based) describing the forward mapping. The i-th element of FWDmap should contain the index of the prototype that the i-th data vector is mapped to.
`nW`	the total number of prototypes in the vector quantizer
`stat`	the statistic to compute. Possible values are • count • sum • mean • sd • q0 (min) • q25 (first quantile) • q50 (median) • q75 (third quantile) • q100 (max)
`parallel`	optional, whether to compute in parallel. Default = TRUE

Details

The statistics are computed individually for each data dimension. The statistics for any prototypes in the vector quantizer whose Receptive Fields are empty (no data mapped to them) are returned as NA. The rows of the returned matrix of statistics represent prototype-level summary statistics (row1 contains stats of prototype1, etc.).

Value

a matrix (nrows = nW, ncol = num. data dimensions) containing the requested statistic for each prototype

summarystat_RF_revmap *Receptive Field Summary Statistics*

Description

Compute summary statistics of each Receptive Field of a vector quantizer, given its reverse mapping. A reverse mapping is a list containing the data indices in the Receptive Field of each prototype.

Usage

```
summarystat_RF_revmap(X_vals, revmap, stat = "mean", parallel = TRUE)
```

Arguments

X_vals	the (possibly multivariate) data values to be summarized, as a matrix (nrows = nrow(X), ncol = num. dimensions)
revmap	a list (length = nW) of indices (1-based) describing the reverse mapping. The i-th element of REVmap should contain the indices of data vectors that are mapped to the the i-th prototype of the quantizer.
stat	the statistic to compute. Possible values are • count • sum • mean • sd • q0 (min) • q25 (first quantile) • q50 (median) • q75 (third quantile) • q100 (max)
parallel	optional, whether to compute in parallel. Default = TRUE

Details

The statistics are computed individually for each data dimension. The statistics for any prototypes in the vector quantizer whose Receptive Fields are empty (no data mapped to them) are returned as NA. The rows of the returned matrix of statistics represent prototype-level summary statistics (row1 contains stats of prototype1, etc.).

Value

a matrix (nrows = nW, ncol = num. data dimensions) containing the requested statistic for each prototype

`tableau20` *Tableau 20 Color Palette*

Description

Tableau 20 Color Palette

Usage

```
tableau20(col = NULL)
```

Value

a named vector containing Tableau 20 colors

`tile_interior_point` *Get a list of interior points in each lattice tile*

Description

Handy for visualization, this function returns the (x,y) coordinates of a point inside each lattice tile that is along a ray of angle `theta` emanating from the tile center. The "radius" of a tile is the distance to the tile boundary in direction `theta`. The (x,y) coordinates are of distance = `rprop` x radius from the tile center.

Usage

```
SOMobj$tile_interior_point(theta, rprop)
```

Arguments

`theta`	the angle at which the (x,y) coordinates should be placed, relative to the tile center
`rprop`	the proportion of the tile radius along angle `theta` at which the (x,y) coordinates will be placed. Should be < 1 to produce a point inside the lattice tile.

Value

a matrix (nrow = nW, ncol = 2) containing the (x,y) coordinates of interior points for each tile. The rows are in neuron-order.

train_SOM *Training function for a SOM object*

Description

(C)SOM training is performed and all associated learning products updated internally, according to DeSieno's algorithm, using the learning rates specified in set_LRAS.

Usage

```
SOMobj$train_SOM(nsteps, X)
```

Arguments

nsteps	the number of training steps to perform
X	the training data matrix. This should be the same matrix that was input to initialize_SOM (checks will be performed on its statistics, if they do not match an error will be returned).

Details

This function calls many of the helper function of the SOM class internally (e.g., update_p, update_W, map_to_netrng, update_learning_rates).

Training is performed online (not batch) according to a sequential random sampling of the rows of `X`. Reproducibility of training results can be achieved by calling R's set.seed function prior to calling `train_SOM`, which will control the random seed used for sampling `X`. The training order used is during each call to `train_SOM` is stored in the field `train_order`.

Additionally, recall_SOM is called at the end of training to update the SOM products.

Parallel computation for BMU selection and prototype updating can be achieved by calling set_parallel prior to calling `train_SOM`.

Various visualizations exist to examine the results of SOM training. See any of the `vis_som_*` functions for more information.

Value

None, SOM products are updated internally

References

DeSieno D (1988). "Adding a conscience to competitive learning." In *IEEE international conference on neural networks*, volume 1(6), 117–124. IEEE Piscataway, NJ.

update_learning_rates *Update the effective learning rates*

Description

The CSOM learning rates (described in set_LRAS) should be annealed over time. This function queries the LRAS given the current training age of the SOM and updates the effective alpha, beta, gamma, sigma, and eta (the last via a call to calc_eta) parameters accordingly.

It is usually not necessary to directly call this function as it is invoked inside train_SOM.

Usage

```
SOMobj$update_learning_rates()
```

Value

None, the above learning rate fields are set internally.

update_p *Update the CSOM win frequencies*

Description

The win frequency field p is updated internally according to DeSieno's CSOM algorithm.

It is usually not necessary to directly call this function as it is invoked inside train_SOM.

Usage

```
SOMobj$update_p(winner_idx)
```

Arguments

winner_idx the prototype index of the BMU selected during a CSOM training step

Value

None

update_W *Update the SOM prototypes*

Description

The prototype matrix W is updated internally according to DeSieno's CSOM algorithm.

It is usually not necessary to directly call this function as it is invoked inside train_SOM.

Usage

```
SOMobj$update_p(winner_idx)
```

Arguments

winner_idx	the prototype index of the BMU selected during a CSOM training step

Value

None

vis_ctab *View a color table*

Description

This is a helper function to provide a quick visualization of the labels and their associated colors that are defined in an input color table.

Usage

```
vis_ctab(ctab, label.cex = 0.9, label.font = 2, nrows_in_display = NULL)
```

Arguments

ctab	the color table dataframe to view. Must have fields `$class`, `$R`, `$G`, `$B`.
label.cex	the size of labels, as plotted on the visual color grid
label.font	the font size of the label, 1=regular, 2 = bold
nrows_in_display	sets the number of rows in the visualized color table. Default = NULL invokes automatic generation of this value, with an attempt to make the resulting color grid as square as possible.

Value

nothing, only used for visualization

vis_LRAS *Visualize the Learning Rate Annealing Schedule*

Description

Visualize the Learning Rate Annealing Schedule

Usage

```
vis_LRAS(SOM)
```

Arguments

SOM	a SOM object

vis_som_annotate *Annotate each tile of the SOM lattice*

Description

Annotate each tile of the SOM lattice

Usage

```
vis_som_annotate(
  SOM,
  add = FALSE,
  text,
  text.cex = 1,
  text.col = "auto",
  text.font = 1,
  text.lightness = 0.4,
  theta = 135,
  rprop = 0,
  active = T,
  subset = NULL,
  change.par = TRUE
)
```

Arguments

SOM	a SOM object
add	whether to create a new SOM visualization panel (=FALSE, default), or add to an existing one (=TRUE)
text	vector of strings used to label each lattice tile
text.cex	the text size used for plotting the labels
text.col	the color used for plotting the labels. Can be one of: • a single color name, which will be recycled across all lattice tiles

- a vector of color names to be applied to each tile
- the reserved keyword "auto", which will set a text color of either white or black, depending on the Lightness (from the HSL colorspace) values of any existing tile fill colors. This is useful if, for example, some tiles are dark (where black text would not show up well) and others are light (where light text would not show up well).

`text.font`	the font used for plotting the labels
`text.lightness`	the Lightness threshold used to determine whether the text.col in each tile is white or black. Only valid if text.col = "auto", default = 0.4. Higher thresholds create more white labels.
`theta`	the angle (in degrees) which specifies the direction of offset of each label, relative to each tile center.
`rprop`	the length of the offset of each label, relative to each tile center. This should be given as a proportion of the total distance from the center to the boundary of each tile, in the direction theta (i.e., `rprop` <= 1)
`active`	Optional, if the SOM object has been recalled, restricts plotting only to active neurons (those whose RF_size > 0). Default = TRUE.
`subset`	Optional, a vector of neuron indices to restrict the plotting to. Default = NULL imposes no restriction (plots whole lattice)
`change.par`	whether to allow the vis function to optimally change the `par` of the plot. Default = TRUE. If `par` is allowed to changed internally, it is always reset upon function exit.

`vis_som_CADJvis` *CADJvis Visualization*

Description

CADJvis Visualization

Usage

```
vis_som_CADJvis(
  SOM,
  TRN,
  add = F,
  nu.pch = 16,
  nu.cex = 1,
  nu.col = "black",
  edge.lwd_range = c(1, 5),
  active = T,
  subset = NULL
)
```

Arguments

`SOM`	a SOM object
`TRN`	a TRN object from package `TopoRNet`

add	whether to create a new plotting device (=FALSE, default), or add to an existing one (=TRUE)
nu.pch	see vis_som_neurons
nu.cex	see vis_som_neurons
nu.col	see vis_som_neurons
edge.lwd_range	the min/max range of the plotted CONN edges, Default = c(1, 5).
active	Optional, if the SOM object has been recalled, restricts plotting (of vertices and edges) only to active neurons (those whose RF_size > 0). Default = TRUE.
subset	Optional, a vector of neuron indices to restrict the plotting to. Default = NULL imposes no restriction (plots whole lattice)

Details

The CADJvis visualization is documented in TopoRNet::vis_CADJvis.

References

Taşdemir K, Merényi E (2009). "Exploiting Data Topology in Visualization and Clustering of Self-Organizing Maps." *IEEE Transactions on Neural Networks*, **20**(4), 549-562. ISSN 1045-9227, doi: 10.1109/TNN.2008.2005409.

---

vis_som_CONNvis *CONNvis Visualization*

Description

CONNvis Visualization

Usage

```
vis_som_CONNvis(
  SOM,
  TRN,
  add = F,
  nu.pch = 16,
  nu.cex = 1,
  nu.col = "black",
  edge.lwd_range = c(1, 5),
  active = T,
  subset = NULL
)
```

Arguments

SOM	a SOM object
TRN	a TRN object from package TopoRNet
add	whether to create a new plotting device (=FALSE, default), or add to an existing one (=TRUE)
nu.pch	see vis_som_neurons

nu.cex	see vis_som_neurons
nu.col	see vis_som_neurons
edge.lwd_range	the min/max range of the plotted CONN edges, Default = c(1, 5).
active	Optional, if the SOM object has been recalled, restricts plotting (of vertices and edges) only to active neurons (those whose RF_size > 0). Default = TRUE.
subset	Optional, a vector of neuron indices to restrict the plotting to. Default = NULL imposes no restriction (plots whole lattice)

Details

The CONNvis visualization is documented in TopoRNet::vis_CONNvis.

References

Taşdemir K, Merényi E (2009). "Exploiting Data Topology in Visualization and Clustering of Self-Organizing Maps." *IEEE Transactions on Neural Networks*, **20**(4), 549-562. ISSN 1045-9227, doi: 10.1109/TNN.2008.2005409.

---

vis_som_fences *Visualize SOM fences*

Description

Visualize SOM fences

Usage

```
vis_som_fences(
  SOM,
  add = FALSE,
  fence.lwd = 2,
  clamp = "none",
  clamp.range = c(0, 1),
  color.mapping = "linear",
  color.palette = c("black", "white"),
  color.range = c(0, 1),
  color.nbins = "none",
  active = T,
  subset = NULL,
  change.par = TRUE
)
```

Arguments

SOM	a SOM object
add	whether to create a new SOM visualization panel (=FALSE, default), or add to an existing one (=TRUE)
fence.lwd	fence line width

- `clamp` controls how values are clamped before unit mapping. Options are: "none" (default), meaning no clamping is used; "identity", meaning the clamp.range is used directly; "quantile", meaning the clamp.range quantiles are used.
- `clamp.range` sets the clamping range. If clamp = 'none', the min/max of values is used. If clamp = 'identity', clamp.range should be given as an effective [lo,hi] range to clamp values to. If clamp = 'quantile', clamp.range should contain the effective [lo,hi] probabilities whose quantiles define the clamp range. Default = c(0,1), which should be changed if clamp = 'identity'.
- `color.mapping` the mapping function used to map values to the range [0,1], which is needed for plotting. Default ='linear' for a linear mapping. Can also be 'zcdf', which first computes the z-score of the values, then maps the z-score to [0,1] via the Std. Normal CDF.
- `color.palette` the color palette used to represent the fence values. This should be given as a character vector of color names which will set the colors which represent the the min (first element) and max (last element) of the fence values. Colors for intermediate values between min and max will be interpolated via colorRamp. Should have at least two elements, but can have more. In this case, the colors will be interpolated throughout the range of those found in `color.palette`. Example: `c('black','yellow','white')` will map the min values to black, the mid values to yellow and the max values to white. Default = c('black','white').
- `color.range` a range in [0,1] defining what portion of the color scale is visualized. This is useful if you want to restrict the plotted color range to the darker or lighter end of the spectrum. Default = c(0,1)
- `color.nbins` allows binning of the (unit-mapped) values prior to color assignment. Options are: 'none' (default), which just maps the values directly to the color range without binning; 'auto', which computes a histogram of the (unit-mapped) values and uses the resulting bin counts; or an integer giving the desired number of bins.
- `active` Optional, if the SOM object has been recalled, restricts plotting only to active neurons (those whose RF_size > 0). Default = TRUE.
- `subset` Optional, a vector of neuron indices to restrict the plotting to. Default = NULL imposes no restriction (plots whole lattice)
- `change.par` whether to allow the vis function to optimally change the `par` of the plot. Default = TRUE. If `par` is allowed to changed internally, it is always reset upon function exit.

`vis_som_gradient` *Visualize a color gradient on SOM lattice tiles*

Description

Visualize a color gradient on SOM lattice tiles

Usage

```
vis_som_gradient(
  SOM,
  add = FALSE,
```

```
    values,
    clamp = "none",
    clamp.range = c(0, 1),
    color.mapping = "linear",
    color.palette = c("black", SOMDisco::tableau20("lightblue")),
    color.range = c(0, 1),
    color.nbins = "none",
    color.NA = "black",
    active = T,
    subset = NULL,
    change.par = TRUE
  )
```

Arguments

SOM	a SOM object
add	whether to create a new SOM visualization panel (=FALSE, default), or add to an existing one (=TRUE)
values	the numeric values used for heatmap coloring of tiles
clamp	controls how values are clamped before unit mapping. Options are: "none" (default), meaning no clamping is used; "identity", meaning the clamp.range is used directly; "quantile", meaning the clamp.range quantiles are used.
clamp.range	sets the clamping range. If clamp = 'none', the min/max of values is used. If clamp = 'identity', clamp.range should be given as an effective [lo,hi] range to clamp values to. If clamp = 'quantile', clamp.range should contain the effective [lo,hi] probabilities whose quantiles define the clamp range. Default = c(0,1), which should be changed if clamp = 'identity'.
color.mapping	the mapping function used to map values to the range [0,1], which is needed for plotting. Default ='linear' for a linear mapping. Can also be 'zcdf', which first computes the z-score of the values, then maps the z-score to [0,1] via the Std. Normal CDF.
color.palette	the color palette used to represent values. This should be given as a character vector of color names which will set the colors which represent the the min (first element) and max (last element) of the `values`. Colors for intermediate values between min and max will be interpolated via colorRamp. Should have at least two elements, but can have more. In this case, the colors will be interpolated throughout the range of those found in `color.palette`. Example: `c('black','lightblue','blue')` will map the min values to black, the mid values to lightblue and the max values to blue. Default = c('black',tableau20('lightblue')).
color.range	a range in [0,1] defining what portion of the color scale is visualized. This is useful if you want to restrict the plotted color range to the darker or lighter end of the spectrum. Default = c(0,1)
color.nbins	allows binning of the (unit-mapped) values prior to color assignment. Options are: • 'none' (default), which maps the values directly to the color range without binning; • 'auto', which computes a histogram of the (unit-mapped) values and uses the resulting bins and counts; • an integer giving the desired number of bins.
color.NA	specifies the color used for any NA values

`active`	Optional, if the SOM object has been recalled, restricts plotting only to active neurons (those whose RF_size > 0). Default = TRUE.
`subset`	Optional, a vector of neuron indices to restrict the plotting to. Default = NULL imposes no restriction (plots whole lattice)
`change.par`	whether to allow the vis function to optimally change the `par` of the plot. Default = TRUE. If `par` is allowed to changed internally, it is always reset upon function exit.

Details

This is a wrapper function for `vis_som_tiles`. The tile colors are deduced from the given values and mapping arguments, then passed along internally.

`vis_som_label` *Visualize the labels of each RF*

Description

If the training data is labeled then the SOM prototypes (and their neurons and RFs) will inherit a label from the learned mapping. Each prototype is labeled by a plurality vote of the labels of data mapped to their RFs.

Usage

```
vis_som_label(
  SOM,
  add = FALSE,
  text.cex = 1,
  text.font = 1,
  theta = 90,
  rprop = 0.75,
  subset = NULL,
  change.par = TRUE
)
```

Arguments

`SOM`	a SOM object
`add`	whether to create a new SOM visualization panel (=FALSE, default), or add to an existing one (=TRUE)
`subset`	Optional, a vector of neuron indices to restrict the plotting to. Default = NULL imposes no restriction (plots whole lattice)
`change.par`	whether to allow the vis function to optimally change the `par` of the plot. Default = TRUE. If `par` is allowed to changed internally, it is always reset upon function exit.

Details

This is a wrapped function which calls `vis_som_tiles` internally using the internal color table `$ctab` to map the `$RF_labels` to colors. Call `$set_ctab` to change this mapping.

`vis_som_labeldist` *Visualize the distribution of labels in each RF*

Description

The distribution of data labels mapped to each Receptive Field is visualized as a proportional pie chart residing in each lattice tile. This function is helpful for determining the organization of SOM neurons: assuming the data labels accurately reflect differences within the dataset, prototypes whose receptive fields contain data of the same label are considered to be better representative than those whose data member labels are mixed.

Usage

```
vis_som_labeldist(SOM, add = FALSE, subset = NULL, change.par = TRUE)
```

Arguments

`SOM`	a SOM object
`add`	whether to create a new SOM visualization panel (=FALSE, default), or add to an existing one (=TRUE)
`subset`	Optional, a vector of neuron indices to restrict the plotting to. Default = NULL imposes no restriction (plots whole lattice)
`change.par`	whether to allow the vis function to optimally change the `par` of the plot. Default = TRUE. If `par` is allowed to changed internally, it is always reset upon function exit.

Details

The internal color table `$ctab` is used to map labels to colors. Call `$set_ctab` to change this mapping.

`vis_som_mUMatrix` *Visualize a color gradient on SOM lattice tiles*

Description

Visualize a color gradient on SOM lattice tiles

Usage

```
vis_som_mUMatrix(
  SOM,
  add = FALSE,
  grad.clamp = "quantile",
  grad.clamp.range = c(0.05, 0.95),
  grad.color.mapping = "linear",
  grad.color.palette = c("black", SOMDisco::tableau20("lightblue")),
  grad.color.range = c(0, 1),
  grad.color.nbins = "auto",
```

```
  fence.lwd = 2,
  fence.clamp = "quantile",
  fence.clamp.range = c(0.05, 0.95),
  fence.color.mapping = "linear",
  fence.color.palette = c("black", "white"),
  fence.color.range = c(0, 1),
  fence.color.nbins = "auto",
  subset = NULL,
  change.par = TRUE
)
```

Arguments

SOM	a SOM object
add	whether to create a new SOM visualization panel (=FALSE, default), or add to an existing one (=TRUE)
grad.clamp	see vis_som_gradient
grad.clamp.range	see vis_som_gradient
grad.color.mapping	see vis_som_gradient
grad.color.palette	see vis_som_gradient
grad.color.range	see vis_som_gradient
grad.color.nbins	see vis_som_gradient
fence.lwd	see vis_som_fences
fence.clamp	see vis_som_fences
fence.clamp.range	see vis_som_fences
fence.color.mapping	see vis_som_fences
fence.color.palette	see vis_som_fences
fence.color.range	see vis_som_fences
fence.color.nbins	see vis_som_fences
subset	Optional, a vector of neuron indices to restrict the plotting to. Default = NULL imposes no restriction (plots whole lattice)
change.par	whether to allow the vis function to optimally change the par of the plot. Default = TRUE. If par is allowed to changed internally, it is always reset upon function exit.

Details

This is a wrapper function which layers vis_som_fences (called with the given fence.* parameters) atop vis_som_gradient (called with the given grad.* parameters).

References

Ultsch A, Siemon HP (1990). "Kohonen's Self Organizing Feature Maps for Exploratory Data Analysis." In Widrow B, Angeniol B (eds.), *Proceedings of the International Neural Network Conference (INNC-90), Paris, France, July 9–13, 1990 1. Dordrecht, Netherlands*, volume 1, 305–308. http://www.uni-marburg.de/fb12/datenbionik/pdf/pubs/1990/UltschSiemon90. Merényi E, Jain A, Villmann T (2007). "Explicit Magnification Control of Self-Organizing Maps for "Forbidden" Data." *IEEE Transactions on Neural Networks*, **18**(3), 786-797.

`vis_som_neurons` *Visualize SOM neurons on the lattice*

Description

Visualize SOM neurons on the lattice

Usage

```
vis_som_neurons(
  SOM,
  add = FALSE,
  nu.pch = 16,
  nu.cex = 1,
  nu.col = "black",
  active = TRUE,
  subset = NULL,
  change.par = TRUE
)
```

Arguments

`SOM`	a SOM object
`add`	whether to create a new SOM visualization panel (=FALSE, default), or add to an existing one (=TRUE)
`nu.pch`	neuron marker type, default = 16
`nu.cex`	neuron marker size, default = 1,
`nu.col`	neuron marker color, default = "black"
`active`	Optional, if the SOM object has been recalled, restricts plotting only to active neurons (those whose RF_size > 0). Default = TRUE.
`subset`	Optional, a vector of neuron indices to restrict the plotting to. Default = NULL imposes no restriction (plots whole lattice)
`change.par`	whether to allow the vis function to optimally change the `par` of the plot. Default = TRUE. If `par` is allowed to changed internally, it is always reset upon function exit.

Details

See ?points or ?text for help with plotting parameters cex, col, font, etc `vis_som_prototypes` *Visualize SOM prototypes at their lattice locations*

Description

Visualize SOM prototypes at their lattice locations

Usage

```
vis_som_prototypes(
  SOM,
  add = FALSE,
  netrng = "int",
  gutter = 0.05,
  wgt.lwd = 1,
  wgt.col = tableau20("blue"),
  axes = T,
  axes.lwd = 0.5,
  axes.col = "black",
  axes.nticksx = 4,
  axes.nticksy = 4,
  active = T,
  subset = NULL,
  change.par = TRUE
)
```

Arguments

`SOM`	a SOM object
`add`	whether to create a new SOM visualization panel (=FALSE, default), or add to an existing one (=TRUE)
`netrng`	whether to plot the prototypes in internal ("int") or external ("ext") network range. Default = "int".
`gutter`	the proportion of the tile width / height used for plotting axes in each tile. The actual space occupied by the axes = gutter * min(tile_width, tile_height)
`wgt.lwd`	the line width used for plotting the weight vectors
`wgt.col`	the color used for plotting the weight vectors
`axes`	boolean, whether to plot individual axes inside each lattice tile
`axes.lwd`	the line width used for plotting axes
`axes.col`	the color used for plotting axes
`axes.nticksx`	the number of ticks on x-axis inside each lattice tile
`axes.nticksy`	the number of ticks on y-axis inside each lattice tile
`active`	Optional, if the SOM object has been recalled, restricts plotting only to active neurons (those whose RF_size > 0). Default = TRUE.
`subset`	Optional, a vector of neuron indices to restrict the plotting to. Default = NULL imposes no restriction (plots whole lattice)
`change.par`	whether to allow the vis function to optimally change the `par` of the plot. Default = TRUE. If `par` is allowed to changed internally, it is always reset upon function exit.

vis_som_setup — *Setup the SOM lattice for visualizations*

Description

Setup the SOM lattice for visualizations

Usage

```
vis_som_setup(
  SOM,
  mar = 0.1,
  lattice_coords = F,
  coords.cex = 0.75,
  active = T,
  subset = NULL,
  change.par = TRUE
)
```

Arguments

SOM	a SOM object
mar	Optional, the plot margin around the lattice (will be recycled for all sides). Default = 0.1
lattice_coords	whether to add the lattice (i,j) coordinates to the left & bottom of the plot
coords.cex	size of lattice coordinate text, if requested
active	Optional, if the SOM object has been recalled, restricts plotting only to active neurons (those whose RF_size > 0). Default = TRUE.
subset	Optional, a vector of neuron indices to restrict the plotting to. Default = NULL imposes no restriction (plots whole lattice)
change.par	whether to allow the vis function to optimally change the `par` of the plot. Default = TRUE. If `par` is allowed to changed internally, it is always reset upon function exit.

vis_som_tiles — *Visualize SOM lattice tiles*

Description

Visualize SOM lattice tiles

Usage

```
vis_som_tiles(
  SOM,
  add = FALSE,
  border.lwd = 1,
  border.lty = 1,
  border.col = "black",
  fill = NULL,
  active = TRUE,
  subset = NULL,
  change.par = TRUE
)
```

Arguments

SOM	a SOM object
add	whether to create a new SOM visualization panel (=FALSE, default), or add to an existing one (=TRUE)
border.lwd	tile border line width, default = 1. Set = 0 to suppress plotting borders.
border.lty	tile border line type, default = 1
border.col	tile border color, default = "black". If border.col = 'fill' and input fill is given, borders will inherit fill colors.
fill	Optional, controls fill color of plotted tiles. If given as a single color name (e.g., "black"), all tiles will be colored with this color. If given as a vector of color names with length = SOM$nW, each tile will be colored separately. Default = NULL means no fill
active	Optional, if the SOM object has been recalled, restricts plotting only to active neurons (those whose RF_size > 0). Default = TRUE.
subset	Optional, a vector of neuron indices to restrict the plotting to. Default = NULL imposes no restriction (plots whole lattice)
change.par	whether to allow the vis function to optimally change the `par` of the plot. Default = TRUE. If `par` is allowed to changed internally, it is always reset upon function exit.

---

vis_som_TopoView *TopoView Visualization*

Description

TopoView Visualization

Usage

```
vis_som_TopoView(
  SOM,
  ADJ,
  add = F,
  nu.pch = 16,
```

```
  nu.cex = 1,
  nu.col = "black",
  edge.col = "darkorange",
  edge.lwd_range = c(1, 5),
  active = T,
  subset = NULL
)
```

Arguments

SOM	a SOM object
ADJ	an adjacency matrix defining edges connecting SOM neurons (or prototypes)
add	whether to create a new plotting device (=FALSE, default), or add to an existing one (=TRUE)
nu.pch	see vis_som_neurons
nu.cex	see vis_som_neurons
nu.col	see vis_som_neurons
edge.lwd_range	the min/max range of the plotted CONN edges, Default = c(1, 5).
active	Optional, if the SOM object has been recalled, restricts plotting (of vertices and edges) only to active neurons (those whose RF_size > 0). Default = TRUE.
subset	Optional, a vector of neuron indices to restrict the plotting to. Default = NULL imposes no restriction (plots whole lattice)
edge.color	line color of plotted edges, default = "darkorange".

Details

The TopoView visualization is documented in `TopoRNet::vis_TopoView`. If the input `ADJ` is weighted and `edge.lwd_range` spans a non-empty set, the visualized edge widths will represent the edge weights (larger weights = thicker edges).

References

Merényi E, Tasdemir K, Zhang L (2009). "Learning Highly Structured Manifolds: Harnessing the Power of SOMs." In Biehl M, Hammer B, Verleysen M, Villmann T (eds.), *Similarity-Based Clustering*, volume 5400 of *Lecture Notes in Computer Science*, 138–168. Springer Verlag, Berlin Heidelberg.

---

vis_som_training *Visualize monitoring measure of SOM training*

Description

Visualize monitoring measure of SOM training

Usage

```
vis_som_training(SOM, vis.SOM = T, vis.mtr = T)
```

Arguments

SOM	a SOM object
vis.SOM	boolean, whether to show the SOM with RF_size gradient, default = T
vis.mtr	boolean, whether to show the monitoring history, if available. Default = T

Index

∗ **datasets**
SHGR, 26

as_list, 3

build_CADJ, 3
build_ctab, 4

calc_Entropy, 4
calc_eta, 5, *34*
calc_SOM_fences, 5
colorRamp, *40, 41*
CONN, 6, *16*

default_LRAS, 7, *19*, *27*
distmat, 7

find_BMU, 8
find_RF_label, 10
find_RF_members, 10

geodesicdist, 11
get_LRAS, 11

infoRDS, *17*
initialize_SOM, 12, *15, 16*, *18*, *20–23*, *25*, *33*

linscale, 12
load, 13, *17*
load_list, 14

map_from_netrng, 14
map_to_netrng, *14*, 15, *33*

new, 15

readRDS, *13*
recall_SOM, *4*, *6*, 16, *18*, *33*

save, *13*, 17
set.seed, *25*, *33*
set_ctab, 17
set_lattice, *12*, 18
set_lattice_fences, *16*, 18
set_LRAS, *5*, *11, 12*, 19, *33, 34*
set_monitoring_freq, 20
set_nBMU, 21
set_netrng, *12*, 21
set_p, 22
set_p_equal, *12*, 23
set_parallel, 22, *33*
set_train_order, 23
set_W, 24
set_W_runif, *12*, *24*, 25
set_X_label, 25
setThreadOptions, *23*
SHGR, 26
SOM, 26
summarystat_RF_fwdmap, 30
summarystat_RF_revmap, 31

tableau20, 32
tile_interior_point, 32
train_SOM, *5*, *15, 16*, *23*, 33, *34, 35*

update_learning_rates, *33*, 34
update_p, *33*, 34
update_W, *33*, 35

vis_ctab, 35
vis_LRAS, 36
vis_som_annotate, 36
vis_som_CADJvis, 37
vis_som_CONNvis, 38
vis_som_fences, *18*, 39
vis_som_gradient, 40
vis_som_label, 42
vis_som_labeldist, 43
vis_som_mUMatrix, 43
vis_som_neurons, 45
vis_som_prototypes, 46
vis_som_setup, 47
vis_som_tiles, 47
vis_som_TopoView, 48
vis_som_training, *20*, 49