sTrainSeq

July 21, 2015

`sTrainSeq` *Function to implement training via sequential algorithm*

Description

`sTrainSeq` is supposed to perform sequential training algorithm. It requires three inputs: a "sMap" or "sInit" object, input data, and a "sTrain" object specifying training environment. The training is implemented iteratively, each training cycle consisting of: i) randomly choose one input vector; ii) determine the winner hexagon/rectangle (BMH) according to minimum distance of codebook matrix to the input vector; ii) update the codebook matrix of the BMH and its neighbors via updating formula (see "Note" below for details). It also returns an object of class "sMap".

Usage

```
sTrainSeq(sMap, data, sTrain, verbose = T)
```

Arguments

`sMap`	an object of class "sMap" or "sInit"
`data`	a data frame or matrix of input data
`sTrain`	an object of class "sTrain"
`verbose`	logical to indicate whether the messages will be displayed in the screen. By default, it sets to TRUE for display

Value

an object of class "sMap", a list with following components:

- `nHex`: the total number of hexagons/rectanges in the grid
- `xdim`: x-dimension of the grid
- `ydim`: y-dimension of the grid
- `lattice`: the grid lattice
- `shape`: the grid shape
- `coord`: a matrix of nHex x 2, with each row corresponding to the coordinates of a hexagon/rectangle in the 2D map grid
- `init`: an initialisation method

- `neighKernel`: the training neighborhood kernel
- `codebook`: a codebook matrix of nHex x ncol(data), with each row corresponding to a prototype vector in input high-dimensional space
- `call`: the call that produced this result

Note

Updating formula is: $m_i(t+1) = m_i(t) + \alpha(t) * h_{wi}(t) * [x(t) - m_i(t)]$, where

- t denotes the training time/step
- i and w stand for the hexagon/rectangle i and the winner BMH w, respectively
- $x(t)$ is an input vector randomly choosen (from the input data) at time t
- $m_i(t)$ and $m_i(t+1)$ are respectively the prototype vectors of the hexagon i at time t and $t+1$
- $\alpha(t)$ is the learning rate at time t. There are three types of learning rate functions:
  - For "linear" function, $\alpha(t) = \alpha_0 * (1 - t/T)$
  - For "power" function, $\alpha(t) = \alpha_0 * (0.005/\alpha_0)^{t/T}$
  - For "invert" function, $\alpha(t) = \alpha_0/(1 + 100 * t/T)$
  - Where α_0 is the initial learing rate (typically, $\alpha_0 = 0.5$ at "rough" stage, $\alpha_0 = 0.05$ at "finetune" stage), T is the length of training time/step (often being set to input data length, i.e., the total number of rows)
- $h_{wi}(t)$ is the neighborhood kernel, a non-increasing function of i) the distance d_{wi} between the hexagon/rectangle i and the winner BMH w, and ii) the radius δ_t at time t. There are five kernels available:
  - For "gaussian" kernel, $h_{wi}(t) = e^{-d_{wi}^2/(2*\delta_t^2)}$
  - For "cutguassian" kernel, $h_{wi}(t) = e^{-d_{wi}^2/(2*\delta_t^2)} * (d_{wi} \le \delta_t)$
  - For "bubble" kernel, $h_{wi}(t) = (d_{wi} \le \delta_t)$
  - For "ep" kernel, $h_{wi}(t) = (1 - d_{wi}^2/\delta_t^2) * (d_{wi} \le \delta_t)$
  - For "gamma" kernel, $h_{wi}(t) = 1/\Gamma(d_{wi}^2/(4 * \delta_t^2) + 2)$

See Also

`sTrainology`, `visKernels`

Examples

```
# 1) generate an iid normal random matrix of 100x10
data <- matrix( rnorm(100*10,mean=0,sd=1), nrow=100, ncol=10)

# 2) from this input matrix, determine nHex=5*sqrt(nrow(data))=50,
# but it returns nHex=61, via "sHexGrid(nHex=50)", to make sure a supra-hexagonal grid
sTopol <- sTopology(data=data, lattice="hexa", shape="suprahex")

# 3) initialise the codebook matrix using "uniform" method
sI <- sInitial(data=data, sTopol=sTopol, init="uniform")

# 4) define trainology at "rough" stage
sT_rough <- sTrainology(sMap=sI, data=data, algorithm="sequential",
stage="rough")

# 5) training at "rough" stage
sM_rough <- sTrainSeq(sMap=sI, data=data, sTrain=sT_rough)
```

```
# 6) define trainology at "finetune" stage
sT_finetune <- sTrainology(sMap=sI, data=data, algorithm="sequential",
stage="finetune")

# 7) training at "finetune" stage
sM_finetune <- sTrainSeq(sMap=sM_rough, data=data, sTrain=sT_rough)
```