sTrainBatch

March 9, 2015

---

sTrainBatch   *Function to implement training via batch algorithm*

Description

sTrainBatch is supposed to perform batch 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, but instead of choosing a single input vector, the whole input matrix is used. In each training cycle, the whole input matrix first land in the map through identifying the corresponding winner hexagon/rectangle (BMH), and then the codebook matrix is updated via updating formula (see "Note" below for details). It returns an object of class "sMap".

Usage

```
sTrainBatch(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 proto-type vector in input high-dimensional space
- call: the call that produced this result

Note

Updating formula is: $m_i(t+1) = \frac{\sum_{j=1}^{dlen} h_{wi}(t)x_j}{\sum_{j=1}^{dlen} h_{wi}(t)}$, where

- t denotes the training time/step
- x_j is an input vector j from the input data matrix (with $dlen$ rows in total)
- i and w stand for the hexagon/rectangle i and the winner BMH w, respectively
- $m_i(t+1)$ is the prototype vector of the hexagon i at time $t+1$
- $h_{wi}(t)$ 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, $h_{wi}(t) = e^{-d_{wi}^2/(2*\delta_t^2)} * (d_{wi} \leq \delta_t)$
  - For "bubble" kernel, $h_{wi}(t) = (d_{wi} \leq \delta_t)$
  - For "ep" kernel, $h_{wi}(t) = (1 - d_{wi}^2/\delta_t^2) * (d_{wi} \leq \delta_t)$
  - 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, stage="rough")

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

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

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