sPipeline

March 27, 2017

`sPipeline`	*Function to setup the pipeline for completing ab initio training given the input data*

Description

`sPipeline` is supposed to finish ab inito training for the input data. It returns an object of class "sMap".

Usage

```
sPipeline(data = NULL, xdim = NULL, ydim = NULL, nHex = NULL,
lattice = c("hexa", "rect"), shape = c("suprahex", "sheet", "triangle",
"diamond", "hourglass", "trefoil", "ladder", "butterfly", "ring",
"bridge"),
init = c("linear", "uniform", "sample"), algorithm = c("batch",
"sequential"), alphaType = c("invert", "linear", "power"),
neighKernel = c("gaussian", "bubble", "cutgaussian", "ep", "gamma"),
finetuneSustain = FALSE, verbose = TRUE)
```

Arguments

`data`	a data frame or matrix of input data
`xdim`	an integer specifying x-dimension of the grid
`ydim`	an integer specifying y-dimension of the grid
`nHex`	the number of hexagons/rectangles in the grid
`lattice`	the grid lattice, either "hexa" for a hexagon or "rect" for a rectangle
`shape`	the grid shape, either "suprahex" for a supra-hexagonal grid or "sheet" for a hexagonal/rectangle sheet. Also supported are suprahex's variants (including "triangle" for the triangle-shaped variant, "diamond" for the diamond-shaped variant, "hourglass" for the hourglass-shaped variant, "trefoil" for the trefoil-shaped variant, "ladder" for the ladder-shaped variant, "butterfly" for the butterfly-shaped variant, "ring" for the ring-shaped variant, and "bridge" for the bridge-shaped variant)
`init`	an initialisation method. It can be one of "uniform", "sample" and "linear" initialisation methods

`algorithm`	the training algorithm. It can be one of "sequential" and "batch" algorithm. By default, it uses 'batch' algorithm purely because of its fast computations (probably also without the compromise of accuracy). However, it is highly recommended not to use 'batch' algorithm if the input data contain lots of zeros; it is because matrix multiplication used in the 'batch' algorithm can be problematic in this context. If much computation resource is at hand, it is alwasy safe to use the 'sequential' algorithm
`alphaType`	the alpha type. It can be one of "invert", "linear" and "power" alpha types
`neighKernel`	the training neighborhood kernel. It can be one of "gaussian", "bubble", "cutgaussian", "ep" and "gamma" kernels
`finetuneSustain`	logical to indicate whether sustain the "finetune" training. If true, it will repeat the "finetune" stage until the mean quantization error does get worse. By default, it sets to true
`verbose`	logical to indicate whether the messages will be displayed in the screen. By default, it sets to false for no 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
- `r`: the hypothetical radius of the grid
- `lattice`: the grid lattice
- `shape`: the grid shape
- `coord`: a matrix of nHex x 2, with rows corresponding to the coordinates of all hexagons/rectangles in the 2D map grid
- `polygon`: a data frame of three columns ('x','y','id') storing polygon location per hexagon in the 2D map grid
- `init`: an initialisation method
- `neighKernel`: the training neighborhood kernel
- `codebook`: a codebook matrix of nHex x ncol(data), with rows corresponding to prototype vectors in input high-dimensional space
- `hits`: a vector of nHex, each element meaning that a hexagon/rectangle contains the number of input data vectors being hit wherein
- `mqe`: the mean quantization error for the "best" BMH
- `call`: the call that produced this result

Note

The pipeline sequentially consists of:

- i) `sTopology` used to define the topology of a grid (with "suprahex" shape by default) according to the input data;
- ii) `sInitial` used to initialise the codebook matrix given the pre-defined topology and the input data (by default using "uniform" initialisation method);

- iii) sTrainology and sTrainSeq used to get the grid map trained at both "rough" and "finetune" stages. If instructed, sustain the "finetune" training until the mean quantization error does get worse;
- iv) sBMH used to identify the best-matching hexagons/rectangles (BMH) for the input data, and these response data are appended to the resulting object of "sMap" class.

References

Hai Fang and Julian Gough. (2014) supraHex: an R/Bioconductor package for tabular omics data analysis using a supra-hexagonal map. *Biochemical and Biophysical Research Communications*, 443(1), 285-289.

See Also

sTopology, sInitial, sTrainology, sTrainSeq, sTrainBatch, sBMH, visHexMulComp

Examples

```
# 1) generate an iid normal random matrix of 100x10
data <- matrix( rnorm(100*10,mean=0,sd=1), nrow=100, ncol=10)
colnames(data) <- paste(rep('S',10), seq(1:10), sep="")

# 2) get trained using by default setup but with different neighborhood kernels
# 2a) with "gaussian" kernel
sMap <- sPipeline(data=data, neighKernel="gaussian")
# 2b) with "bubble" kernel
# sMap <- sPipeline(data=data, neighKernel="bubble")
# 2c) with "cutgaussian" kernel
# sMap <- sPipeline(data=data, neighKernel="cutgaussian")
# 2d) with "ep" kernel
# sMap <- sPipeline(data=data, neighKernel="ep")
# 2e) with "gamma" kernel
# sMap <- sPipeline(data=data, neighKernel="gamma")

# 3) visualise multiple component planes of a supra-hexagonal grid
visHexMulComp(sMap, colormap="jet", ncolors=20, zlim=c(-1,1),
gp=grid::gpar(cex=0.8))

# 4) get trained using by default setup but using the shape "butterfly"
sMap <- sPipeline(data=data, shape="trefoil",
algorithm=c("batch","sequential")[2])
visHexMulComp(sMap, colormap="jet", ncolors=20, zlim=c(-1,1),
gp=grid::gpar(cex=0.8))
```