sCompReorder

September 23, 2015

`sCompReorder` *Function to reorder component planes*

Description

`sCompReorder` is supposed to reorder component planes for the input map/data. It returns an object of class "sReorder". It is realized by using a new map grid (with sheet shape consisting of a rectangular lattice) to train component plane vectors (either column-wise vectors of codebook/data matrix or the covariance matrix thereof). As a result, similar component planes are placed closer to each other. It is highly recommend to use trained map (i.e. codebook matrix) as input if data matrix is hugely big to save computational costs.

Usage

```
sCompReorder(sMap, xdim = NULL, ydim = NULL, amplifier = NULL,
metric = c("none", "pearson", "spearman", "kendall", "euclidean",
"manhattan", "cos", "mi"), init = c("linear", "uniform", "sample"),
algorithm = c("sequential", "batch"), alphaType = c("invert", "linear",
"power"), neighKernel = c("gaussian", "bubble", "cutgaussian", "ep",
"gamma"))
```

Arguments

`sMap`	an object of class "sMap" or input data frame/matrix
`xdim`	an integer specifying x-dimension of the grid
`ydim`	an integer specifying y-dimension of the grid
`amplifier`	an integer specifying the amplifier (3 by default) of the number of component planes. The product of the component number and the amplifier constitutes the number of rectangles in the sheet grid
`metric`	distance metric used to difine the similarity between component planes. It can be "none", which means directly using column-wise vectors of codebook/data matrix. Otherwise, first calculate the covariance matrix from the codebook/data matrix. The distance metric used for calculating the covariance matrix between component planes can be: "pearson" for pearson correlation, "spearman" for spearman rank correlation, "kendall" for kendall tau rank correlation, "euclidean" for euclidean distance, "manhattan" for cityblock distance, "cos" for cosine similarity, "mi" for mutual information. See `sDistance` for details

`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 'sequential' algorithm. If the input data contains a large number of samples but not a great amount of zero entries, then it is reasonable to use 'batch' algorithm for its fast computations (probably also without the compromise of accuracy)
`alphaType`	the alpha type. It can be one of "invert", "linear" and "power" alpha types
`neighKernel`	the training neighbor kernel. It can be one of "gaussian", "bubble", "cutgaussian", "ep" and "gamma" kernels

Value

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

- `nHex`: the total number of rectanges in the grid
- `xdim`: x-dimension of the grid
- `ydim`: y-dimension of the grid
- `uOrder`: the unique order/placement for each component plane that is reordered to the "sheet"-shape grid with rectangular lattice
- `coord`: a matrix of nHex x 2, with each row corresponding to the coordinates of each "uOrder" rectangle in the 2D map grid
- `call`: the call that produced this result

Note

All component planes are uniquely placed within a "sheet"-shape rectangle grid:

- Each component plane mapped to the "sheet"-shape grid with rectangular lattice is determinied iteratively in an order from the best matched to the next compromised one.
- If multiple compoments are hit in the same rectangular lattice, the worse one is always sacrificed by moving to the next best one till all components are placed somewhere exclusively on their own.

The size of "sheet"-shape rectangle grid depends on the input arguments:

- How the input parameters are used to determine nHex is taken priority in the following order: "xdim & ydim" > "nHex" > "data".
- If both of xdim and ydim are given, $nHex = xdim * ydim$.
- If only data is input, $nHex = 5 * sqrt(dlen)$, where dlen is the number of rows of the input data.
- After nHex is determined, xy-dimensions of rectangle grid are then determined according to the square root of the two biggest eigenvalues of the input data.

See Also

`sTopology`, `sPipeline`, `sBMH`, `sDistance`, `visCompReorder`

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
sMap <- sPipeline(data=data)

# 3) reorder component planes in different ways
# 3a) directly using column-wise vectors of codebook matrix
sReorder <- sCompReorder(sMap=sMap, amplifier=2, metric="none")
# 3b) according to covariance matrix of pearson correlation of codebook matrix
sReorder <- sCompReorder(sMap=sMap, amplifier=2, metric="pearson")
# 3c) according to covariance matrix of pearson correlation of input matrix
sReorder <- sCompReorder(sMap=data, amplifier=2, metric="pearson")
```