Package 'clusterExperiment'

February 22, 2021

Title Compare Clusterings for Single-Cell Sequencing

Version 2.10.1

Description Provides functionality for running and comparing many different clusterings of single-cell sequencing data or other large mRNA Expression data sets.

BugReports https://github.com/epurdom/clusterExperiment/issues

License Artistic-2.0

Encoding UTF-8

Depends R (>= 3.6.0), SingleCellExperiment, SummarizedExperiment (>= 1.15.4), BiocGenerics

Imports methods, NMF, RColorBrewer, ape (>= 5.0), cluster, stats, limma, howmany, locfdr, matrixStats, graphics, parallel, RSpectra, kernlab, stringr, S4Vectors, grDevices, DelayedArray (>= 0.7.48), HDF5Array (>= 1.7.10), Matrix, Rcpp, edgeR, scales, zinbwave, phylobase, pracma, mbkmeans

Suggests BiocStyle, knitr, testthat, MAST, Rtsne, scran, igraph

VignetteBuilder knitr

LazyData false

LazyLoad false

RoxygenNote 7.1.0

biocViews Clustering, RNASeq, Sequencing, Software, SingleCell

Collate 'AllChecks.R' 'AllClassesCE.R' 'AllClassesCF.R' 'AllGenerics.R' 'AllHelper.R' 'AllHelperClusterFunction.R' 'AllHelperDendro.R' 'AllHelperFilter.R' 'JiashinJiCode.R' 'RcppExports.R' 'addClusterings.R' 'assignUnassigned.R' 'subsampleClustering.R' 'internalDendroFunctions.R' 'internalClusterFunctions.R' 'internalFunctions.R' 'builtInClusterFunctions.R' 'clusterContrasts.R' 'clusterMany.R' 'clusterSingle.R' 'dataCreation.R' 'deprecateFunctions.R' 'getClusterIndex.R' 'getFeatures.R' 'getParams.R' 'getReducedData.R' 'mainClustering.R' 'makeBlankData.R' 'makeConsensus.R' 'makeDendrogram.R' 'makeFilterStats.R' 'makeReducedDims.R' 'mergeClusters.R' 'plotBarplot.R' 'plotClusters.R' 'plotClustersTable.R' 'plotClustersWorkflow.R' 'plotContrastHeatmap.R' 'plotDendrogram.R' 'plotFeatureBoxplot.R'

'plotFeatureScatter.R' 'plotHeatmap.R' 'plotReduceDim.R'
'plottingHelpers.R' 'rsec.R' 'seqCluster.R' 'subsampleLoop.R'
'subset.R' 'transformFunction.R' 'updateObject.R'
'workflowClusters.R'

LinkingTo Rcpp

git_url https://git.bioconductor.org/packages/clusterExperiment

git_branch RELEASE_3_12

git_last_commit 499be02

git_last_commit_date 2021-02-05

Date/Publication 2021-02-21

Author Elizabeth Purdom [aut, cre, cph],
Davide Risso [aut]

Maintainer Elizabeth Purdom <epurdom@stat.berkeley.edu>

R topics documented:

`addClusterings` *Add clusterings to ClusterExperiment object*

Description

Function for adding new clusterings in form of vector (single cluster) or matrix (multiple clusterings) to an existing ClusterExperiment object

Usage

```
## S4 method for signature 'ClusterExperiment,matrix'
addClusterings(
  x,
  y,
  clusterTypes = "User",
  clusterLabels = NULL,
  clusterLegend = NULL
)

## S4 method for signature 'ClusterExperiment,ClusterExperiment'
addClusterings(x, y, transferFrom = c("x", "y"), mergeCEObjects = FALSE)

## S4 method for signature 'ClusterExperiment,vector'
addClusterings(x, y, makePrimary = FALSE, ...)
```

Arguments

`x`	a ClusterExperiment object
`y`	additional clusters to add to x. Can be a ClusterExperiment object or a matrix/vector of clusters.
`clusterTypes`	a string describing the nature of the clustering. The values 'clusterSingle', 'clusterMany', 'mergeClusters', 'makeConsensus' are reserved for the clustering coming from the package workflow and should not be used when creating a new object with the constructor.
`clusterLabels`	label(s) for the clusters being added. If `y` a matrix, the column names of that matrix will be used by default, if `clusterLabels` is not given.
`clusterLegend`	a list giving the cluster legend for the clusters added.

transferFrom	If x and y are both ClusterExperiment objects indicates from which object the clustering info should be taken (regarding merging, dendrogram, etc). Does not affect the order of the clusterings, which will always be the clusterings of x, followed by those of y (along with slots 'clusterType', 'clusterInfo', 'clusterLegend')
mergeCEObjects	logical If x and y are both ClusterExperiment objects indicates as to whether should try to grab in the information missing from x from y (or vice versa if transferFrom=y).
makePrimary	whether to make the added cluster the primary cluster (only relevant if y is a vector)
...	For addClusterings, passed to signature ClusterExperiment,matrix. For [(subsetting), passed to SingleCellExperiment subsetting function.

Details

addClusterings adds y to x, and is thus not symmetric in the two arguments. In particular, the primaryCluster, all of the dendrogram information, the merge information, coClustering, and orderSamples are all kept from the x object, even if y is a ClusterExperiment.

Value

A ClusterExperiment object.

Examples

```
data(simData)

cl1 <- clusterSingle(simData, subsample=FALSE,
sequential=FALSE, mainClusterArgs=list(clusterArgs=list(k=3),
clusterFunction="pam"))
cl2 <- clusterSingle(simData, subsample=FALSE,
sequential=FALSE, mainClusterArgs=list(clusterArgs=list(k=3),
clusterFunction="pam"))

addClusterings(cl1, cl2)
```

---

assignUnassigned *Assign unassigned samples to nearest cluster*

Description

Assigns the unassigned samples in a cluster to the nearest cluster based on distance to the medians of the clusters.

Usage

```
## S4 method for signature 'ClusterExperiment'
assignUnassigned(
  object,
  whichCluster = "primary",
  clusterLabel,
```

```
  makePrimary = TRUE,
  whichAssay = 1,
  reduceMethod = "none",
  ...
)

## S4 method for signature 'ClusterExperiment'
removeUnassigned(object, whichCluster = "primary")
```

Arguments

object	A Cluster Experiment object
whichCluster	argument that can be a single numeric or character value indicating the *single* clustering to be used. Giving values that result in more than one clustering will result in an error. See details of getClusterIndex.
clusterLabel	if missing, the current cluster label of the cluster will be appended with the string "_AllAssigned".
makePrimary	whether to make the added cluster the primary cluster (only relevant if y is a vector)
whichAssay	which assay to use to calculate the median per cluster and take dimensionality reduction (if requested)
reduceMethod	character. A method (or methods) for reducing the size of the data, either by filtering the rows (genes) or by a dimensionality reduction method. Must either be 1) must match the name of a built-in method, in which case if it is not already existing in the object will be passed to makeFilterStats or link{makeReducedDims}, or 2) must match a stored filtering statistic or dimensionality reduction in the object
...	arguments passed to getReducedData specifying the dimensionality reduction (if any) to be taken of the data for calculating the medians of the clusters

Details

The function assignUnassigned calculates the median values of each variable for each cluster, and then calculates the euclidean distance of each unassigned sample to the median of each cluster. Each unassigned sample is assigned to the cluster for which it closest to the median.

All unassigned samples in the cluster are given a clustering, regardless of whether they are classified as -1 or -2.

removeUnclustered removes all samples that are unclustered (i.e. -1 or -2 assignment) in the designated cluster of object (so they may be unclustered in other clusters found in clusterMatrix(object)).

Value

The function assignUnassigned returns a ClusterExperiment object with the unassigned samples assigned to one of the existing clusters.

The function removeUnassigned returns a ClusterExperiment object with the unassigned samples removed.

See Also

getReducedData

Examples

```
#load CE object
data(rsecFluidigm)
smallCE<-rsecFluidigm[,1:50]
#assign the unassigned samples
assignUnassigned(smallCE, makePrimary=TRUE)

#note how samples are REMOVED:
removeUnassigned(smallCE)
```

`clusterContrasts` *Create contrasts for testing DE of a cluster*

Description

Uses clustering to create different types of contrasts to be tested that can then be fed into DE testing programs.

Usage

```
## S4 method for signature 'ClusterExperiment'
clusterContrasts(cluster, contrastType, ...)

## S4 method for signature 'vector'
clusterContrasts(
  cluster,
  contrastType = c("Dendro", "Pairs", "OneAgainstAll"),
  dendro = NULL,
  pairMat = NULL,
  outputType = c("limma", "MAST"),
  removeUnassigned = TRUE
)
```

Arguments

`cluster`	Either a vector giving contrasts assignments or a ClusterExperiment object
`contrastType`	What type of contrast to create. 'Dendro' traverses the given dendrogram and does contrasts of the samples in each side, 'Pairs' does pair-wise contrasts based on the pairs given in pairMat (if pairMat=NULL, does all pairwise), and 'OneAgainstAll' compares each cluster to the average of all others.
`...`	arguments that are passed to from the `ClusterExperiment` version to the most basic numeric version.
`dendro`	The dendrogram to traverse if contrastType="Dendro". Note that this should be the dendrogram of the clusters, not of the individual samples, either of class "dendrogram" or "phylo4"
`pairMat`	matrix giving the pairs of clusters for which to do pair-wise contrasts (must match to elements of cl). If NULL, will do all pairwise of the clusters in `cluster` (excluding "-1" categories). Each row is a pair to be compared and must match the names of the clusters in the vector `cluster`.

`outputType` character string. Gives format for the resulting contrast matrix. Currently the two options are the format appropriate for `limma` and `MAST` package.

`removeUnassigned` logical, whether to remove negative valued clusters from the design matrix. Appropriate to pick TRUE (default) if design will be input into linear model on samples that excludes -1.

Details

The input vector must be numeric clusters, but the external commands that make the contrast matrix (e.g. `makeContrasts`) require syntatically valid R names. For this reason, the names of the levels will be "X1" instead of "1". And negative values (if removeUnassigned=FALSE) will be "X.1","X.2", etc.

Value

List with components:

- `contrastMatrix` Contrast matrix, the form of which depends on `outputType`. If `outputType=="limma"`, the result of running `makeContrasts`: a matrix with number of columns equal to the number of contrasts, and rows equal to the number of levels of the factor that will be fit in a linear model.
- `contrastNames`A vector of names for each of the contrasts. NULL if no such additional names.

Author(s)

Elizabeth Purdom

References

Ritchie, ME, Phipson, B, Wu, D, Hu, Y, Law, CW, Shi, W, and Smyth, GK (2015). limma powers differential expression analyses for RNA-sequencing and microarray studies. Nucleic Acids Research 43, e47. http://nar.oxfordjournals.org/content/43/7/e47

Finak, et al. MAST: a flexible statistical framework for assessing transcriptional changes and characterizing heterogeneity in single-cell RNA sequencing data. Genome Biology (2015).

Examples

```
data(simData)
cl <- clusterMany(simData,nReducedDims=c(5,10,50),
reduceMethod="PCA", makeMissingDiss=TRUE,
clusterFunction="pam", ks=2:4, findBestK=c(FALSE), removeSil=TRUE,
subsample=FALSE)
#Pairs:
clusterContrasts(cl,contrastType="Pairs")
#Dendrogram
cl<-makeDendrogram(cl)
clusterContrasts(cl,contrastType="Pairs")
```

clusterDendrogram *Accessing and manipulating the dendrograms*

Description

These functions are for accessing and manipulating the dendrograms stored in a ClusterExperiment object. We also document the required format of these dendrograms here.

Usage

```
## S4 method for signature 'ClusterExperiment'
clusterDendrogram(x)

## S4 method for signature 'ClusterExperiment'
sampleDendrogram(x)

## S4 method for signature 'ClusterExperiment'
nInternalNodes(x)

## S4 method for signature 'ClusterExperiment'
nTips(x)

## S4 method for signature 'ClusterExperiment'
nNodes(x)

## S4 replacement method for signature 'ClusterExperiment'
nodeLabels(x, ...) <- value

## S4 method for signature 'ClusterExperiment,phylo4d,phylo4d'
checkDendrogram(x, dendroCluster, dendroSample, whichCluster = "dendro")

## S4 method for signature 'ClusterExperiment'
nodeLabels(x)

## S4 method for signature 'ClusterExperiment'
nodeIds(x, type = c("all", "internal", "tip"))

## S4 method for signature 'ClusterExperiment'
convertToDendrogram(x)
```

Arguments

x	a ClusterExperiment object
...	additional options passed to nodeLabels<- (ignored)
value	replacement value for nodeLabels. See details.
dendroCluster	a phylo4d to be check as for being cluster hierarchy
dendroSample	a phylo4d to be check as for being cluster hierarchy
whichCluster	argument that can be a single numeric or character value indicating the *single* clustering to be used. Giving values that result in more than one clustering will result in an error. See details of getClusterIndex.
type	the type of node to return results from. One of "all", "internal", and "tip".

Details

Two dendrograms are stored in a ClusterExperiment object. One is a dendrogram that describes the hierarchy between the clusters (@dendro_clusters), and the other is a dendrogram that extends that hierarchy to include the clusters (@dendro_samples). The clustering that is used to make these hierarchies is saved in as well (@dendro_index)

The dendrograms stored in a ClusterExperiment object are required to be a phylo4d-class from the package phylobase (which uses the basic format of the S3 class phylo in the ape package to store the edges; phylobase makes it a S4 class with some useful helpers). This class allows storage of a data.frame of information corresponding to information on each node (see tdata).

Additional requirements are made of these dendrograms to be a valid for the slots of the ClusterExperiment class, described below, regarding the data that must be stored with it and the labels which can be assigned. Possible dendrograms can be checked for validity with the function checkDendrogram. The reason for the restrictions on the labels is so as to not duplicate storage of the names, see below descriptions for where to save user-defined names.

- LabelsThe cluster dendrogram can only have labels on the *internal* nodes. Labels on the internal nodes of the cluster dendrogram can be set by the user (the function nodeLabels<- is defined to work on a ClusterExperiment object to make this easy). The tips of the cluster dendrogram, corresponding to the clusters, cannot have labels; users can set the labels (e.g. for plotting, etc) in the clusterLegend slot using the function renameClusters.
- Data The cluster hierarchy must have data stored with it that has the following columns (additional ones are allowed):
  - NodeIdThe permanent node id for the node. Must be of the format "NodeIdX" where "X" is a integer.
  - PositionThe type of node, in terms of its position. The internal nodes should have the values "cluster hierarchy node" while the tips should have "cluster hierarchy tip".
  - ClusterIdDendroOnly for tips of dendrogram, should have the id that corresponds to its cluster in the clustering of the @dendro_index. Of the form "ClusterIdX", where "X" is the internal cluster id (see clusterLegend). Internal nodes should have NA values.
  - ClusterIdMergeThe id that corresponds to the cluster in the clustering of the @merge_index, if it exists. Of the form "ClusterIdX", where "X" is the internal cluster id (see clusterLegend

- LabelsThe sample dendrogram is not allowed to have ANY labels. The names for those nodes that correspond to the cluster hierarchy will be pulled from the names in the cluster hierarchy for plotting, etc. and should be set there (see above). Sample names for the tips of the tree will be pulled from colnames of the object and should be set there.
- Data The cluster hierarchy must have data stored with it that has the following columns (additional ones are allowed):
  - NodeIdFor those nodes that correspond to a node in the cluster hierarchy, should have its permanent node id in this column. Other nodes should be NA.
  - PositionThe type of node, in terms of its position. The internal nodes should have the values "cluster hierarchy node" while the tips should have "cluster hierarchy tip".
  - SampleIndexOnly for tips of dendrogram, the index of the sample at that tip to the samples in the object.

For setting the node labels of the cluster dendrogram via nodeLabels<-, the replacement value has to have names that match the internal ids of the cluster dendrogram (the NodeId column).

Value

`clusterDendrogram` returns the dendrogram describing the clustering hierarchy.

`sampleDendrogram` returns the dendrogram that expands the cluster hierarchy to the samples.

`nInternalNodes` returns the number of *internal* nodes of the cluster hierarchy.

`nTips` returns the number of tips of the cluster hierarchy (same as number of non-negative clusters in the dendrogram clustering)

`nNodes` returns the number of total nodes of the cluster hierarchy

`nodeLabels<-` sets the node labels of the *cluster* dendrogram

`checkClusterDendrogram` checks if a `phylo4d` objects are valid for the cluster and sample dendrogram slots of the given `ClusterExperiment` object. Returns `TRUE` if there are no problems. Otherwise creates error.

`nodeLabels` returns the node labels of the *cluster* dendrogram

`nodeIds` returns the internal (permanent) node ids of the *cluster* dendrogram

`convertToDendrogram` returns the sample dendrogram converted to a `dendrogram` class.

The Stored Dendrograms

NA

Cluster Hierarchy

NA

Sample Hierarchy

NA

Helper Functions

NA

See Also

`makeDendrogram`, `phylo4d-class`, `phylo`

`dendrogram`

Examples

```
data(rsecFluidigm)
# retrieve the dendrogram of the clusters:
head(clusterDendrogram(rsecFluidigm),5)
# retrieve the dendrogram of the samples:
head(sampleDendrogram(rsecFluidigm),5)
# Return # internal nodes from cluster hierarchy
nInternalNodes(rsecFluidigm)
# Return # tips from cluster hierarchy (i.e. # clusters)
nTips(rsecFluidigm)
# Return internal node ids
nodeIds(rsecFluidigm,type="internal")
# Labels assigned to internal nodes
nodeLabels(rsecFluidigm)
```

```
# Assign new labels to the internal nodes of the cluster hierarchy
l1<-paste("A", 1:nInternalNodes(rsecFluidigm),sep=":")
names(l1)<-nodeIds(rsecFluidigm,type="internal")
nodeLabels(rsecFluidigm)<-l1
nodeLabels(rsecFluidigm)
```

`ClusterExperiment-class` *Class ClusterExperiment*

Description

`ClusterExperiment` is a class that extends `SingleCellExperiment` and is used to store the data and clustering information.

In addition to the slots of the `SingleCellExperiment` class, the `ClusterExperiment` object has the additional slots described in the Slots section.

There are several methods implemented for this class. The most important methods (e.g., `clusterMany`, `makeConsensus`, ...) have their own help page. Simple helper methods are described in the Methods section. For a comprehensive list of methods specific to this class see the Reference Manual.

The constructor `ClusterExperiment` creates an object of the class `ClusterExperiment`. However, the typical way of creating these objects is the result of a call to `clusterMany` or `clusterSingle`.

Note that when subsetting the data, the co-clustering and dendrogram information are lost.

Usage

```
ClusterExperiment(object, clusters, ...)

## S4 method for signature 'matrixOrHDF5,ANY'
ClusterExperiment(object, clusters, ...)

## S4 method for signature 'SummarizedExperiment,ANY'
ClusterExperiment(object, clusters, ...)

## S4 method for signature 'SingleCellExperiment,numeric'
ClusterExperiment(object, clusters, ...)

## S4 method for signature 'SingleCellExperiment,character'
ClusterExperiment(object, clusters, ...)

## S4 method for signature 'SingleCellExperiment,factor'
ClusterExperiment(object, clusters, ...)

## S4 method for signature 'SingleCellExperiment,matrix'
ClusterExperiment(
  object,
  clusters,
  transformation = function(x) {     x },
  primaryIndex = 1,
  clusterTypes = "User",
  clusterInfo = NULL,
```

```
  orderSamples = seq_len(ncol(object)),
  dendro_samples = NULL,
  dendro_index = NA_real_,
  dendro_clusters = NULL,
  coClustering = NULL,
  merge_index = NA_real_,
  merge_cutoff = NA_real_,
  merge_dendrocluster_index = NA_real_,
  merge_nodeProp = NULL,
  merge_nodeMerge = NULL,
  merge_method = NA_character_,
  merge_demethod = NA_character_,
  clusterLegend = NULL,
  checkTransformAndAssay = TRUE
)
```

Arguments

object	a matrix or SummarizedExperiment or SingleCellExperiment containing the data that was clustered.
clusters	can be either a numeric or character vector, a factor, or a numeric matrix, containing the cluster labels.
...	The arguments transformation, clusterTypes and clusterInfo to be passed to the constructor for signature SingleCellExperiment,matrix.
transformation	function. A function to transform the data before performing steps that assume normal-like data (i.e. constant variance), such as the log.
primaryIndex	integer. Sets the 'primaryIndex' slot (see Slots).
clusterTypes	a string describing the nature of the clustering. The values 'clusterSingle', 'clusterMany', 'mergeClusters', 'makeConsensus' are reserved for the clustering coming from the package workflow and should not be used when creating a new object with the constructor.
clusterInfo	a list with information on the clustering (see Slots).
orderSamples	a vector of integers. Sets the 'orderSamples' slot (see Slots).
dendro_samples	phylo4 object. Sets the 'dendro_samples' slot (see Slots).
dendro_index	numeric. Sets the dendro_index slot (see Slots).
dendro_clusters	phylo4 object. Sets the 'dendro_clusters' slot (see Slots).
coClustering	matrix. Sets the coClustering slot (see Slots).
merge_index	integer. Sets the merge_index slot (see Slots)
merge_cutoff	numeric. Sets the merge_cutoff slot (see Slots)
merge_dendrocluster_index	integer. Sets the merge_dendrocluster_index slot (see Slots)
merge_nodeProp	data.frame. Sets the merge_nodeProp slot (see Slots)
merge_nodeMerge	data.frame. Sets the merge_nodeMerge slot (see Slots)
merge_method	character, Sets the merge_method slot (see Slots)
merge_demethod	character, Sets the merge_demethod slot (see Slots)

`clusterLegend` list, Sets the `clusterLegend` slot (see details).

`checkTransformAndAssay`
logical. Whether to check the content of the assay and given transformation function for whether they are valid.

Details

The `clusterLegend` argument to `ClusterExperiment` must be a valid clusterLegend format and match the values in `clusters`, in that the "clusterIds" column must matches the value in the clustering matrix `clusters`. If `names(clusterLegend)==NULL`, it is assumed that the entries of `clusterLegend` are in the same order as the columns of `clusters`. Generally, this is not a good way for users to set the clusterLegend slot.

The `ClusterExperiment` constructor function gives clusterLabels based on the column names of the input matrix/SingleCellExperiment. If missing, will assign labels "cluster1","cluster2", etc.

Note that the validity check when creating a new `ClusterExperiment` object with `new` is less extensive than when using `ClusterExperiment` function with `checkTransformAndAssay=TRUE` (the default). Users are advised to use `ClusterExperiment` to create new `ClusterExperiment` objects.

Value

A `ClusterExperiment` object.

Slots

`transformation` function. Function to transform the data by when methods that assume normal-like data (e.g. log)

`clusterMatrix` matrix. A matrix giving the integer-valued cluster ids for each sample. The rows of the matrix correspond to clusterings and columns to samples. The integer values are assigned in the order that the clusters were found, if found by setting sequential=TRUE in clusterSingle. "-1" indicates the sample was not clustered.

`primaryIndex` numeric. An index that specifies the primary set of labels.

`clusterInfo` list. A list with info about the clustering. If created from `clusterSingle`, clusterInfo will include the parameter used for the call, and the call itself. If `sequential = TRUE` it will also include the following components.

- `clusterInfo`if sequential=TRUE and clusters were successfully found, a matrix of information regarding the algorithm behavior for each cluster (the starting and stopping K for each cluster, and the number of iterations for each cluster).
- `whyStop`if sequential=TRUE and clusters were successfully found, a character string explaining what triggered the algorithm to stop.

`merge_index` index of the current merged cluster

`merge_cutoff` value for the cutoff used to determine whether to merge clusters

`merge_dendrocluster_index` index of the cluster merged with the current merge

`merge_nodeMerge` data.frame of information about nodes merged in the current merge. See `mergeClusters`

`merge_nodeProp` data.frame of information of proportion estimated non-null at each node of dendrogram. See `mergeClusters`

`merge_method` character indicating method used for merging. See `mergeClusters`

`merge_demethod` character indicating the DE method used for merging. See `mergeClusters`

`clusterTypes` character vector with the origin of each column of clusterMatrix.

dendro_samples phylo4d object. A dendrogram containing the cluster relationship (leaves are samples; see clusterDendrogram for details).

dendro_clusters phylo4d object. A dendrogram containing the cluster relationship (leaves are clusters; see see sampleDendrogram for details).

dendro_index numeric. An integer giving the cluster that was used to make the dendrograms. NA_real_ value if no dendrograms are saved.

coClustering One of

- NULL, i.e. empty
- a numeric vector, signifying the indices of the clusterings in the clusterMatrix that were used for makeConsensus. This allows for the recreation of the distance matrix (using hamming distance) if needed for function plotClusters but doesn't require storage of full NxN matrix.
- a sparseMatrix object – a sparse representation of the NxN matrix with the cluster co-occurrence information; this can either be based on subsampling or on co-clustering across parameter sets (see clusterMany). The matrix is a square matrix with number of rows/columns equal to the number of samples.

clusterLegend a list, one per cluster in clusterMatrix. Each element of the list is a matrix with nrows equal to the number of different clusters in the clustering, and consisting of at least two columns with the following column names: "clusterId" and "color".

orderSamples a numeric vector (of integers) defining the order of samples to be used for plotting of samples. Usually set internally by other functions.

See Also

sparseMatrix phylo4d

Examples

```
sce <- matrix(data=rnorm(200), ncol=10)
labels <- gl(5, 2)

cc <- ClusterExperiment(sce, as.numeric(labels), transformation =
function(x){x})
```

---

clusterExperiment-deprecated

Deprecated functions in package 'clusterExperiment'

Description

These functions are provided for compatibility with older versions of 'clusterExperiment' only, and will be defunct at the next release.

Usage

```
## S4 method for signature 'ANY'
combineMany(x, ...)
```

Arguments

x	any object
...	additional arguments

Details

The following functions are deprecated and will be made defunct; use the replacement indicated below:

- combineMany: makeConsensus
- removeUnclustered: removeUnassigned

ClusterExperiment-methods

Helper methods for the ClusterExperiment class

Description

This is a collection of helper methods for the ClusterExperiment class.

Usage

```
## S4 method for signature 'ClusterExperiment'
show(object)

## S4 method for signature 'ClusterExperiment'
transformation(x)

## S4 replacement method for signature 'ClusterExperiment,`function`'
transformation(object) <- value

## S4 method for signature 'ClusterExperiment'
nClusterings(x)

## S4 method for signature 'ClusterExperiment'
nClusters(x, ignoreUnassigned = TRUE)

## S4 method for signature 'ClusterExperiment'
nFeatures(x)

## S4 method for signature 'ClusterExperiment'
nSamples(x)

## S4 method for signature 'ClusterExperiment'
clusterMatrixNamed(x, whichClusters = "all")

## S4 method for signature 'ClusterExperiment'
clusterMatrixColors(x, whichClusters = "all")

## S4 method for signature 'ClusterExperiment'
```

```
clusterMatrix(x, whichClusters)

## S4 method for signature 'ClusterExperiment'
primaryCluster(x)

## S4 method for signature 'ClusterExperiment'
primaryClusterIndex(x)

## S4 method for signature 'ClusterExperiment'
primaryClusterLabel(x)

## S4 method for signature 'ClusterExperiment'
primaryClusterNamed(x)

## S4 method for signature 'ClusterExperiment'
primaryClusterType(x)

## S4 replacement method for signature 'ClusterExperiment,numeric'
primaryClusterIndex(object) <- value

## S4 method for signature 'ClusterExperiment'
dendroClusterIndex(x)

## S4 method for signature 'ClusterExperiment'
coClustering(x)

## S4 replacement method for signature 'ClusterExperiment,matrix'
coClustering(object) <- value

## S4 replacement method for signature 'ClusterExperiment,dsCMatrix'
coClustering(object) <- value

## S4 replacement method for signature 'ClusterExperiment,numeric'
coClustering(object) <- value

## S4 method for signature 'ClusterExperiment'
clusterTypes(x)

## S4 method for signature 'ClusterExperiment'
clusteringInfo(x)

## S4 method for signature 'ClusterExperiment'
clusterLabels(x)

## S4 replacement method for signature 'ClusterExperiment,character'
clusterLabels(object) <- value

## S4 method for signature 'ClusterExperiment'
clusterLegend(x)

## S4 replacement method for signature 'ClusterExperiment,list'
clusterLegend(object) <- value
```

```
## S4 method for signature 'ClusterExperiment'
orderSamples(x)

## S4 replacement method for signature 'ClusterExperiment,numeric'
orderSamples(object) <- value

## S4 replacement method for signature 'ClusterExperiment,character'
clusterTypes(object) <- value

## S4 method for signature 'ClusterExperiment'
addToColData(object, ...)

## S4 method for signature 'ClusterExperiment'
colDataClusters(
  object,
  whichClusters = "primary",
  useNames = TRUE,
  makeFactor = TRUE,
  ...
)
```

Arguments

x, object	a ClusterExperiment object.
value	The value to be substituted in the corresponding slot. See the slot descriptions in ClusterExperiment for details on what objects may be passed to these functions.
ignoreUnassigned	logical. If true, ignore the clusters with -1 or -2 assignments in calculating the number of clusters per clustering.
whichClusters	argument that can be either numeric or character vector indicating the clusterings to be used. See details of getClusterIndex.
...	For addToColData, arguments passed to colDataClusters.
useNames	for tableClusters, whether the output should be tabled with names (useNames=TRUE) or ids (useNames=FALSE)
makeFactor	logical for colDataClusters. If TRUE the clustering will be added to the colData slot as a factor. If FALSE, the clustering will be added to the colData slot as a character vector if useNames=TRUE and as a numeric vector if useNames=FALSE.

Details

Note that redefining the transformation function via transformation(x)<- will check the validity of the transformation on the data assay. If the assay is large, this may be time consuming. Consider using a call to ClusterExperiment, which has the option as to whether to check the validity of the transformation.

Value

transformation prints the function used to transform the data prior to clustering.

nClusterings returns the number of clusterings (i.e., ncol of clusterMatrix).

`nClusters` returns the number of clusters per clustering

`nFeatures` returns the number of features (same as 'nrow').

`nSamples` returns the number of samples (same as 'ncol').

`clusterMatrixNamed` returns a matrix with cluster labels.

`clusterMatrixColors` returns the matrix with all the clusterings, using the internally stored colors for each cluster

`clusterMatrix` returns the matrix with all the clusterings.

`primaryCluster` returns the primary clustering (as numeric).

`primaryClusterIndex` returns/sets the primary clustering index (i.e., which column of clusterMatrix corresponds to the primary clustering).

`primaryClusterIndex` returns/sets the primary clustering index (i.e., which column of clusterMatrix corresponds to the primary clustering).

`primaryClusterNamed` returns the primary cluster (using cluster labels).

`primaryClusterIndex` returns/sets the primary clustering index (i.e., which column of clusterMatrix corresponds to the primary clustering).

`dendroClusterIndex` returns/sets the clustering index of the clusters used to create dendrogram (i.e., which column of clusterMatrix corresponds to the clustering).

`coClustering` returns/sets the co-clustering matrix.

`clusterTypes` returns/sets the clusterTypes slot.

`clusteringInfo` returns the clusterInfo slot.

`clusterLabels` returns/sets the column names of the clusterMatrix slot.

`clusterLegend` returns/sets the clusterLegend slot.

`orderSamples` returns/sets the orderSamples slot.

`addToColData` returns a `ClusterExperiment` object with the clusterings in clusterMatrix slot added to the `colData` slot

`colDataClusters` returns a `DataFrame` object that has the clusterings in clusterMatrix slot added to the `DataFrame` in the `colData` slot

Examples

```
# load data:
data(rsecFluidigm)
show(rsecFluidigm)
#Number of clusterings
nClusterings(rsecFluidigm)
# Number of clusters per clustering
nClusters(rsecFluidigm)
# Number of features/samples
nSamples(rsecFluidigm)
nFeatures(rsecFluidigm)
# retrieve all clustering assignments
# (either as cluster ids, cluster names or cluster colors)
head(clusterMatrix(rsecFluidigm)[,1:5])
head(clusterMatrixNamed(rsecFluidigm)[,1:5])
head(clusterMatrixColors(rsecFluidigm)[,1:5])
# clustering Types/Labels
clusterTypes(rsecFluidigm)
clusterLabels(rsecFluidigm)
```

```
# Add a clustering assignment to the colData of the object
# (useful if working with function that relies on colData)
colData(rsecFluidigm)
test<-addToColData(rsecFluidigm,whichCluster="primary")
colData(test)
```

ClusterFunction-methods

Helper methods for the ClusterFunction class

Description

This is a collection of helper methods for the ClusterExperiment class.

Usage

```
## S4 method for signature 'character'
requiredArgs(object)

## S4 method for signature 'ClusterFunction'
requiredArgs(object, genericOnly = FALSE)

## S4 method for signature 'list'
requiredArgs(object)

## S4 method for signature 'character'
requiredArgs(object)

## S4 method for signature 'character'
requiredArgs(object)

## S4 method for signature 'factor'
requiredArgs(object)

## S4 method for signature 'ClusterFunction'
algorithmType(object)

## S4 method for signature 'character'
algorithmType(object)

## S4 method for signature 'factor'
algorithmType(object)

## S4 method for signature 'list'
algorithmType(object)

## S4 method for signature 'ClusterFunction'
inputType(object)

## S4 method for signature 'list'
inputType(object)
```

```
## S4 method for signature 'character'
inputType(object)

## S4 method for signature 'factor'
inputType(object)
```

Arguments

object	input to the method, either a ClusterFunction class or a character describing a built-in ClusterFunction object. Can also be a list of ClusterFunction objects, in which case the list must have names for each function.
genericOnly	logical If TRUE, return only the generic required arguments (i.e. those required by the algorithm type) and not the arguments specific to that clustering found in the slot requiredArgs. If FALSE both sets of arguments are returned.

Details

Note that when subsetting the data, the dendrogram information and the co-clustering matrix are lost.

Value

requiredArgs returns a list of the required args of a function (via a call to requiredArgs)

algorithmType returns a character value giving the type of clustering function ("01" or "K")

inputType returns a character value giving the input type of the object

---

clusterMany *Create a matrix of clustering across values of parameters*

Description

Given a range of parameters, this function will return a matrix with the clustering of the samples across the range, which can be passed to plotClusters for visualization.

Usage

```
## S4 method for signature 'matrixOrHDF5'
clusterMany(
  x,
  reduceMethod = "none",
  nReducedDims = NA,
  transFun = NULL,
  isCount = FALSE,
  ...
)

## S4 method for signature 'SingleCellExperiment'
clusterMany(
  x,
  ks = NA,
```

```
  clusterFunction,
  reduceMethod = "none",
  nFilterDims = defaultNDims(x, reduceMethod, type = "filterStats"),
  nReducedDims = defaultNDims(x, reduceMethod, type = "reducedDims"),
  alphas = 0.1,
  findBestK = FALSE,
  sequential = FALSE,
  removeSil = FALSE,
  subsample = FALSE,
  silCutoff = 0,
  distFunction = NA,
  betas = 0.9,
  minSizes = 1,
  transFun = NULL,
  isCount = FALSE,
  verbose = TRUE,
  parameterWarnings = FALSE,
  mainClusterArgs = NULL,
  subsampleArgs = NULL,
  seqArgs = NULL,
  whichAssay = 1,
  makeMissingDiss = if (ncol(x) < 1000) TRUE else FALSE,
  ncores = 1,
  random.seed = NULL,
  run = TRUE,
  ...
)

## S4 method for signature 'ClusterExperiment'
clusterMany(
  x,
  reduceMethod = "none",
  nFilterDims = defaultNDims(x, reduceMethod, type = "filterStats"),
  nReducedDims = defaultNDims(x, reduceMethod, type = "reducedDims"),
  eraseOld = FALSE,
  ...
)

## S4 method for signature 'SummarizedExperiment'
clusterMany(x, ...)

## S4 method for signature 'data.frame'
clusterMany(x, ...)
```

Arguments

x	the data matrix on which to run the clustering. Can be object of the following classes: matrix (with genes in rows), SummarizedExperiment, SingleCellExperiment or ClusterExperiment.
reduceMethod	character A character identifying what type of dimensionality reduction to perform before clustering. Options are 1) "none", 2) one of listBuiltInReducedDims() or listBuiltInFitlerStats OR 3) stored filtering or reducedDim values in

	the object.
nReducedDims	vector of the number of dimensions to use (when reduceMethod gives a dimensionality reduction method).
transFun	a transformation function to be applied to the data. If the transformation applied to the data creates an error or NA values, then the function will throw an error. If object is of class ClusterExperiment, the stored transformation will be used and giving this parameter will result in an error.
isCount	if transFun=NULL, then isCount=TRUE will determine the transformation as defined by function(x){log2(x+1)}, and isCount=FALSE will give a transformation function function(x){x}. Ignored if transFun=NULL. If object is of class ClusterExperiment, the stored transformation will be used and giving this parameter will result in an error.
...	For signature matrix, arguments to be passed on to mclapply (if ncores>1). For all the other signatures, arguments to be passed to the method for signature matrix.
ks	the range of k values (see details for the meaning of k for different choices of other parameters).
clusterFunction	function used for the clustering. This must be either 1) a character vector of built-in clustering techniques, or 2) a *named* list of ClusterFunction objects. Current functions can be found by typing listBuiltInFunctions() into the command-line.
nFilterDims	vector of the number of the most variable features to keep (when "var", "abscv", or "mad" is identified in reduceMethod).
alphas	values of alpha to be tried. Only used for clusterFunctions of type '01'. Determines tightness required in creating clusters from the dissimilarity matrix. Takes on values in [0,1]. See documentation of ClusterFunction.
findBestK	logical, whether should find best K based on average silhouette width (only used when clusterFunction of type "K").
sequential	logical whether to use the sequential strategy (see details of seqCluster). Can be used in combination with subsample=TRUE or FALSE.
removeSil	logical as to whether remove when silhouette < silCutoff (only used if clusterFunction of type "K")
subsample	logical as to whether to subsample via subsampleClustering. If TRUE, clustering in mainClustering step is done on the co-occurance between clusterings in the subsampled clustering results. If FALSE, the mainClustering step will be run directly on x/diss
silCutoff	Requirement on minimum silhouette width to be included in cluster (only for combinations where removeSil=TRUE).
distFunction	a vector of character strings that are the names of distance functions found in the global environment. See the help pages of clusterSingle for details about the required format of distance functions. Currently, this distance function must be applicable for all clusterFunction types tried. Therefore, it is not possible in clusterMany to intermix type "K" and type "01" algorithms if you also give distances to evaluate via distFunction unless all distances give 0-1 values for the distance (and hence are possible for both type "01" and "K" algorithms).
betas	values of beta to be tried in sequential steps. Only used for sequential=TRUE. Determines the similarity between two clusters required in order to deem the cluster stable. Takes on values in [0,1]. See documentation of seqCluster.

`minSizes` the minimimum size required for a cluster (in the `mainClustering` step). Clusters smaller than this are not kept and samples are left unassigned.

`verbose` logical. If TRUE it will print informative messages.

`parameterWarnings` logical, as to whether warnings and comments from checking the validity of the parameter combinations should be printed.

`mainClusterArgs` list of arguments to be passed for the mainClustering step, see help pages of `mainClustering`.

`subsampleArgs` list of arguments to be passed to the subsampling step (if `subsample=TRUE`), see help pages of `subsampleClustering`.

`seqArgs` list of arguments to be passed to `seqCluster`.

`whichAssay` numeric or character specifying which assay to use. See `assay` for details.

`makeMissingDiss` logical. Whether to calculate necessary distance matrices needed when input is not "diss". If TRUE, then when a clustering function calls for a inputType "diss", but the given matrix is of type "X", the function will calculate a distance function. A dissimilarity matrix will also be calculated if a post-processing argument like `findBestK` or `removeSil` is chosen, since these rely on calcualting silhouette widths from distances.

`ncores` the number of threads

`random.seed` a value to set seed before each run of clusterSingle (so that all of the runs are run on the same subsample of the data). Note, if 'random.seed' is set, argument 'ncores' should NOT be passed via subsampleArgs; instead set the argument 'ncores' of clusterMany directly (which is preferred for improving speed anyway).

`run` logical. If FALSE, doesn't run clustering, but just returns matrix of parameters that will be run, for the purpose of inspection by user (with rownames equal to the names of the resulting column names of clMat object that would be returned if `run=TRUE`). Even if `run=FALSE`, however, the function will create the dimensionality reductions of the data indicated by the user input.

`eraseOld` logical. Only relevant if input `x` is of class `ClusterExperiment`. If TRUE, will erase existing workflow results (clusterMany as well as mergeClusters and makeConsensus). If FALSE, existing workflow results will have `"_i"` added to the clusterTypes value, where `i` is one more than the largest such existing workflow clusterTypes.

Details

Some combinations of these parameters are not feasible. See the documentation of `clusterSingle` for important information on how these parameter choices interact.

While the function allows for multiple values of clusterFunction, the code does not reuse the same subsampling matrix and try different clusterFunctions on it. This is because if sequential=TRUE, different subsample clusterFunctions will create different sets of data to subsample so it is not possible; if sequential=FALSE, we have not implemented functionality for this reuse. Setting the `random.seed` value, however, should mean that the subsampled matrix is the same for each, but there is no gain in computational complexity (i.e. each subsampled co-occurence matrix is recalculated for each set of parameters).

The argument `ks` is interpreted differently for different choices of the other parameters. When/if sequential=TRUE, `ks` defines the argument `k0` of `seqCluster`. Otherwise, `ks` values are the `k`

values for **both** the mainClustering and subsampling step (i.e. assigned to the subsampleArgs and mainClusterArgs that are passed to mainClustering and subsampleClustering unless k is set appropriately in subsampleArgs. The passing of these arguments via subsampleArgs will only have an effect if 'subsample=TRUE'. Similarly, the passing of mainClusterArgs[["k"]] will only have an effect when the clusterFunction argument includes a clustering algorithm of type "K". When/if "findBestK=TRUE", ks also defines the kRange argument of mainClustering unless kRange is specified by the user via the mainClusterArgs; note this means that the default option of setting kRange that depends on the input k (see mainClustering) is not available in clusterMany, only in clusterSingle.

If the input is a ClusterExperiment object, current implementation is that existing orderSamples,coClustering or the many dendrogram slots will be retained.

If run=FALSE, the function will still calculate reduced dimensions or filter statistics if not already calculated and saved in the object. Moreover the results of these calculations will not be save. Therefore, if these steps are lengthy for large datasets it is recommended to do them before calling the function.

The given reduceMethod values must either be *all* precalculated filtering/dimensionality reduction stored in the appropriate location, or must *all* be character values giving a built-in filtering/dimensionality reduction methods to be calculated. If some of the filtering/dimensionality methods are already calculated and stored, but not all, then they will *all* be recalculated (and if they are not all built-in methods, this will give an error). So to save computational time with pre-calculated dimensionality reduction, the user must make sure they are *all* precalculated. Also, user-defined values (i.e. not built-in functions) cannot be mixed with built-in functions unless they have already been precalculated (see makeFilterStats or makeReducedDims).

Value

If run=TRUE will return a ClusterExperiment object, where the results are stored as clusterings with clusterTypes clusterMany. Depending on eraseOld argument above, this will either delete existing such objects, or change the clusterTypes of existing objects. See argument eraseOld above. Arbitrarily the first clustering is set as the primaryClusteringIndex.

If run=FALSE a list with elements:

- paramMatrix a matrix giving the parameters of each clustering, where each column is a possible parameter set by the user and passed to clusterSingle and each row of paramMatrix corresponds to a clustering in clMat
- mainClusterArgs a list of (possibly modified) arguments to mainClusterArgs
- seqArgs=seqArgsa list of (possibly modified) arguments to seqArgs
- subsampleArgsa list of (possibly modified) arguments to subsampleArgs

Examples

```
data(simData)

#Example: clustering using pam with different dimensions of pca and different
#k and whether remove negative silhouette values
#check how many and what runs user choices will imply:
checkParams <- clusterMany(simData,reduceMethod="PCA", makeMissingDiss=TRUE,
   nReducedDims=c(5,10,50), clusterFunction="pam", isCount=FALSE,
   ks=2:4,findBestK=c(TRUE,FALSE),removeSil=c(TRUE,FALSE),run=FALSE)
print(head(checkParams$paramMatrix))

#Now actually run it
```

```
cl <- clusterMany(simData,reduceMethod="PCA", nReducedDims=c(5,10,50),  isCount=FALSE,
   clusterFunction="pam",ks=2:4,findBestK=c(TRUE,FALSE),makeMissingDiss=TRUE,
   removeSil=c(TRUE,FALSE))
print(cl)
head(colnames(clusterMatrix(cl)))

#make names shorter for plotting
clNames <- clusterLabels(cl)
clNames <- gsub("TRUE", "T", clNames)
clNames <- gsub("FALSE", "F", clNames)
clNames <- gsub("k=NA,", "", clNames)

par(mar=c(2, 10, 1, 1))
plotClusters(cl, axisLine=-2,clusterLabels=clNames)


## Not run:
#following code takes around 1+ minutes to run because of the subsampling
#that is redone each time:
system.time(clusterTrack <- clusterMany(simData, ks=2:15,
    alphas=c(0.1,0.2,0.3), findBestK=c(TRUE,FALSE), sequential=c(FALSE),
    subsample=c(FALSE), removeSil=c(TRUE), clusterFunction="pam",
    makeMissingDiss=TRUE,
     mainClusterArgs=list(minSize=5, kRange=2:15), ncores=1, random.seed=48120))

## End(Not run)
```

`clusterSingle` *General wrapper method to cluster the data*

Description

Given input data, this function will find clusters, based on a single specification of parameters.

Usage

```
## S4 method for signature 'SummarizedExperiment'
clusterSingle(inputMatrix, ...)

## S4 method for signature 'ClusterExperiment'
clusterSingle(inputMatrix, ...)

## S4 method for signature 'SingleCellExperiment'
clusterSingle(
  inputMatrix,
  reduceMethod = "none",
  nDims = defaultNDims(inputMatrix, reduceMethod),
  whichAssay = 1,
  ...
)

## S4 method for signature 'matrixOrHDF5OrNULL'
```

```
clusterSingle(
  inputMatrix,
  inputType = "X",
  subsample = FALSE,
  sequential = FALSE,
  distFunction = NA,
  mainClusterArgs = NULL,
  subsampleArgs = NULL,
  seqArgs = NULL,
  isCount = FALSE,
  transFun = NULL,
  reduceMethod = "none",
  nDims = defaultNDims(inputMatrix, reduceMethod),
  makeMissingDiss = if (ncol(inputMatrix) < 1000) TRUE else FALSE,
  clusterLabel = "clusterSingle",
  saveSubsamplingMatrix = FALSE,
  checkDiss = FALSE,
  warnings = TRUE
)
```

Arguments

inputMatrix	numerical matrix on which to run the clustering or a SummarizedExperiment, SingleCellExperiment, or ClusterExperiment object.
...	arguments to be passed on to the method for signature matrix.
reduceMethod	character A character identifying what type of dimensionality reduction to perform before clustering. Options are 1) "none", 2) one of listBuiltInReducedDims() or listBuiltInFitlerStats OR 3) stored filtering or reducedDim values in the object.
nDims	integer An integer identifying how many dimensions to reduce to in the reduction specified by reduceMethod. Defaults to output of defaultNDims
whichAssay	numeric or character specifying which assay to use. See assay for details.
inputType	a character vector defining what type of input is given in the inputMatrix argument. Must consist of values "diss","X", or "cat" (see details). "X" and "cat" should be indicate matrices with features in the row and samples in the column; "cat" corresponds to the features being numerical integers corresponding to categories, while "X" are continuous valued features. "diss" corresponds to an inputMatrix that is a NxN dissimilarity matrix. "cat" is largely used internally for clustering of sets of clusterings.
subsample	logical as to whether to subsample via subsampleClustering. If TRUE, clustering in mainClustering step is done on the co-occurance between clusterings in the subsampled clustering results. If FALSE, the mainClustering step will be run directly on x/diss
sequential	logical whether to use the sequential strategy (see details of seqCluster). Can be used in combination with subsample=TRUE or FALSE.
distFunction	a distance function to be applied to inputMatrix. Only relevant if inputType="X". See details of clusterSingle for the required format of the distance function.
mainClusterArgs	list of arguments to be passed for the mainClustering step, see help pages of mainClustering.

subsampleArgs list of arguments to be passed to the subsampling step (if subsample=TRUE), see help pages of subsampleClustering.

seqArgs list of arguments to be passed to seqCluster.

isCount if transFun=NULL, then isCount=TRUE will determine the transformation as defined by function(x){log2(x+1)}, and isCount=FALSE will give a transformation function function(x){x}. Ignored if transFun=NULL. If object is of class ClusterExperiment, the stored transformation will be used and giving this parameter will result in an error.

transFun a transformation function to be applied to the data. If the transformation applied to the data creates an error or NA values, then the function will throw an error. If object is of class ClusterExperiment, the stored transformation will be used and giving this parameter will result in an error.

makeMissingDiss logical. Whether to calculate necessary distance matrices needed when input is not "diss". If TRUE, then when a clustering function calls for a inputType "diss", but the given matrix is of type "X", the function will calculate a distance function. A dissimilarity matrix will also be calculated if a post-processing argument like findBestK or removeSil is chosen, since these rely on calcualting silhouette widths from distances.

clusterLabel a string used to describe the clustering. By default it is equal to "clusterSingle", to indicate that this clustering is the result of a call to clusterSingle.

saveSubsamplingMatrix logical. If TRUE, the co-clustering matrix resulting from subsampling is returned in the coClustering slot (and replaces any existing coClustering object in the slot coClustering if input object is a ClusterExperiment object.)

checkDiss logical. Whether to check whether the dissimilarities matrices are valid (whether given by the user or calculated because makeMissingDiss=TRUE).

warnings logical. Whether to print out the many possible warnings and messages regarding checking the internal consistency of the parameters.

Details

clusterSingle is an 'expert-oriented' function, intended to be used when a user wants to run a single clustering and/or have a great deal of control over the clustering parameters. Most users will find clusterMany more relevant. However, clusterMany makes certain assumptions about the intention of certain combinations of parameters that might not match the user's intent; similarly clusterMany does not directly take a dissimilarity matrix but only a matrix of values x (though a user can define a distance function to be applied to x in clusterMany).

Unlike clusterMany, most of the relevant arguments for the actual clustering algorithms in clusterSingle are passed to the relevant steps via the arguments mainClusterArgs, subsampleArgs, and seqArgs. These arguments should be *named* lists with parameters that match the corresponding functions: mainClustering,subsampleClustering, and seqCluster. These three functions are not meant to be called by the user, but rather accessed via calls to clusterSingle. But the user can look at the help files of those functions for more information regarding the parameters that they take.

Only certain combinations of parameters are possible for certain choices of sequential and subsample. These restrictions are documented below.

- clusterFunction for mainClusterArgs: The choice of subsample=TRUE also controls what algorithm type of clustering functions can be used in the mainClustering step. When subsample=TRUE, then resulting co-clustering matrix from subsampling is converted to a dissimilarity (specificaly 1-coclustering values) and is passed to diss of mainClustering. For this reason, the

ClusterFunction object given to mainClustering via the argument mainClusterArgs must take input of the form of a dissimilarity. When subsample=FALSE and sequential=TRUE, the clusterFunction passed in clusterArgs element of mainClusterArgs must define a ClusterFunction object with algorithmType 'K'. When subsample=FALSE and sequential=FALSE, then there are no restrictions on the ClusterFunction and that clustering is applied directly to the input data.

- clusterFunction for subsampleArgs: If the ClusterFunction object given to the clusterArgs of subsamplingArgs is missing the algorithm will use the default for subsampleClustering (currently "pam"). If sequential=TRUE, this ClusterFunction object must be of type 'K'.
- Setting k for subsampling: If subsample=TRUE and sequential=TRUE, the current K of the sequential iteration determines the 'k' argument passed to subsampleClustering so setting 'k=' in the list given to the subsampleArgs will not do anything and will produce a warning to that effect (see documentation of seqCluster).
- Setting k for mainClustering step: If sequential=TRUE then the user should not set k in the clusterArgs argument of mainClusterArgs because it must be set by the sequential code, which has a iterative reseting of the parameters. Specifically if subsample=FALSE, then the sequential method iterates over choices of k to cluster the input data. And if subsample=TRUE, then the k in the clustering of mainClustering step (assuming the clustering function is of type 'K') will use the k used in the subsampling step to make sure that the k used in the mainClustering step is reasonable.
- Setting findBestK in mainClusterArgs: If sequential=TRUE and subsample=FALSE, the user should not set 'findBestK=TRUE' in mainClusterArgs. This is because in this case the sequential method changes k; an error message will be given if this combination of options are set. However, if sequential=TRUE and subsample=TRUE, then passing either 'findBestK=TRUE' or 'findBestK=FALSE' via mainClusterArgs will function as expected (assuming the clusterFunction argument passed to mainClusterArgs is of type 'K'). In particular, the sequential step will set the number of clusters k for clustering of each subsample. If findBestK=FALSE, that same k will be used for mainClustering step that clusters the resulting co-occurance matrix after subsampling. If findBestK=TRUE, then mainClustering will search for best k. Note that the default 'kRange' over which mainClustering searches when findBestK=TRUE depends on the input value of k which is set by the sequential method if sequential=TRUE), see above. The user can change kRange to not depend on k and to be fixed across all of the sequential steps by setting kRange explicitly in the mainClusterArgs list.

To provide a distance matrix via the argument distFunction, the function must be defined to take the distance of the rows of a matrix (internally, the function will call distFunction(t(x)). This is to be compatible with the input for the dist function. as.matrix will be performed on the output of distFunction, so if the object returned has a as.matrix method that will convert the output into a symmetric matrix of distances, this is fine (for example the class dist for objects returned by dist have such a method). If distFunction=NA, then a default distance will be calculated based on the type of clustering algorithm of clusterFunction. For type "K" the default is to take dist as the distance function. For type "01", the default is to take the (1-cor(x))/2.

Value

A ClusterExperiment object if inputType is of type "X".

If input was not of type "X", then the result is a list with values

- clustering: The vector of clustering results
- clusterInfo: A list with information about the parameters run in the clustering

- coClusterMatrix: (only if saveSubsamplingMatrix=TRUE, NxB set of clusterings obtained after B subsamples.

See Also

clusterMany to compare multiple choices of parameters, and mainClustering,subsampleClustering, and seqCluster for the underlying functions called by clusterSingle.

Examples

```
data(simData)

## Not run:
#following code takes some time.
#use clusterSingle to do sequential clustering
#(same as example in seqCluster only using clusterSingle ...)
set.seed(44261)
clustSeqHier_v2 <- clusterSingle(simData,
     sequential=TRUE, subsample=TRUE,
     subsampleArgs=list(resamp.n=100, samp.p=0.7,
     clusterFunction="kmeans", clusterArgs=list(nstart=10)),
     seqArgs=list(beta=0.8, k0=5), mainClusterArgs=list(minSize=5,
     clusterFunction="hierarchical01",clusterArgs=list(alpha=0.1)))

## End(Not run)

#use clusterSingle to do just clustering k=3 with no subsampling
clustObject <- clusterSingle(simData,
    subsample=FALSE, sequential=FALSE,
    mainClusterArgs=list(clusterFunction="pam", clusterArgs=list(k=3)))
#compare to standard pam
pamOut<-cluster::pam(t(simData),k=3,cluster.only=TRUE)
all(pamOut==primaryCluster(clustObject))
```

---

fluidigmData *Subset of fluidigm data*

Description

Subset of fluidigm data

Format

subset of fluidigm data used in vignette package.

Details

fluidigmData and fluidigmColData are portions of the fluidigm data distributed in the package scRNAseq package. We have subsetted to only the cells sequenced under high depth, and limited our selves to only two of the four gene estimates provided by scRNAseq ("tophat_counts" and "rsem_tpm").

Author(s)

Elizabeth Purdom <epurdom@stat.berkeley.edu>

See Also

fluidigm

Examples

```
#code used to create objects:
## Not run:
library(scRNAseq)
if(packageVersion("scRNAseq")>="1.11.0") fluidigm <- ReprocessedFluidigmData() else data(fluidigm)
fluidSubset<- fluidigm[,colData(fluidigm)[,"Coverage_Type"]=="High"]
fluidigmData<-assays(fluidSubset)[c("tophat_counts","rsem_tpm")]
fluidigmColData<-as.data.frame(colData(fluidSubset))
usethis::use_data(fluidigmData, fluidigmColData, overwrite=FALSE)

## End(Not run)
```

---

getBestFeatures *Function for finding best features associated with clusters*

Description

Calls limma on input data to determine features most associated with found clusters (based on an F-statistic, pairwise comparisons, or following a tree that clusters the clusters).

Usage

```
## S4 method for signature 'matrixOrHDF5'
getBestFeatures(
  x,
  cluster,
  contrastType = c("F", "Dendro", "Pairs", "OneAgainstAll"),
  dendro = NULL,
  pairMat = NULL,
  weights = NULL,
  contrastAdj = c("All", "PerContrast", "AfterF"),
  DEMethod = c("edgeR", "limma", "limma-voom"),
  dgeArgs = NULL,
  ...
)

## S4 method for signature 'ClusterExperiment'
getBestFeatures(
  x,
  contrastType = c("F", "Dendro", "Pairs", "OneAgainstAll"),
  whichCluster = "primary",
  whichAssay = 1,
  DEMethod,
  weights = if ("weights" %in% assayNames(x)) "weights" else NULL,
```

```
  ...
)
```

Arguments

x	data for the test. Can be a numeric matrix or a ClusterExperiment.
cluster	A numeric vector with cluster assignments. "-1" indicates the sample was not assigned to a cluster.
contrastType	What type of test to do. 'F' gives the omnibus F-statistic, 'Dendro' traverses the given dendrogram and does contrasts of the samples in each side, 'Pairs' does pair-wise contrasts based on the pairs given in pairMat (if pairMat=NULL, does all pairwise), and 'OneAgainstAll' compares each cluster to the average of all others. Passed to clusterContrasts
dendro	The dendrogram to traverse if contrastType="Dendro". Note that this should be the dendrogram of the clusters, not of the individual samples, either of class "dendrogram" or "phylo4"
pairMat	matrix giving the pairs of clusters for which to do pair-wise contrasts (must match to elements of cl). If NULL, will do all pairwise of the clusters in cluster (excluding "-1" categories). Each row is a pair to be compared and must match the names of the clusters in the vector cluster.
weights	weights to use in by edgeR. If x is a matrix, then weights should be a matrix of weights, of the same dimensions as x. If x is a ClusterExperiment object weights can be a either a matrix, as previously described, or a character or numeric index to an assay in x that contains the weights. We recommend that weights be stored as an assay with name "weights" so that the weights will also be used with mergeClusters, and this is the default. Setting weights=NULL ensures that weights will NOT be used, and only the standard edgeR.
contrastAdj	What type of FDR correction to do for contrasts tests (i.e. if contrastType='Dendro' or 'Pairs').
DEMethod	character vector describing how the differential expression analysis should be performed (replaces previous argument isCount. See details.
dgeArgs	a list of arguments to pass to DGEList which is the starting point for both edgeR and limma-voom methods of DE. This includes normalization factors/total count values etc.
...	If x is a matrix, these are options to pass to topTable or topTableF (see limma package). If x is a ClusterExperiment object, these arguments can also be those to pass to the matrix version.
whichCluster	argument that can be a single numeric or character value indicating the *single* clustering to be used. Giving values that result in more than one clustering will result in an error. See details of getClusterIndex.
whichAssay	numeric or character specifying which assay to use. See assay for details.

Details

getBestFeatures returns the top ranked features corresponding to a cluster assignment. It uses either limma or edgeR to fit the models, and limma/edgeR functions topTable or topTableF to find the best features. See the options of these functions to put better control on what gets returned (e.g. only if significant, only if log-fc is above a certain amount, etc.). In particular, set 'number=' to define how many significant features to return (where number is per contrast for the 'Pairs' or 'Dendro' option)

DEMethod triggers what type of differential expression analysis will be performed. Three options are available: limma, edgeR, and limma with a voom corrections. The last two options are only appropriate for count data. If the input x is a ClusterExperiment object, and DEMethod="limma", then the data analyzed for DE will be *after* taking the transformation of the data (as given in the transformation slot of the object). For the options "limma-voom" and "edgeR", the transformation slot will be ignored and only the counts data (as specified by the whichAssay slot) will be passed to the programs. Note that for "limma-voom" this implies that the data will be transformed by voom with the function log2(x+0.5). If weights is not NULL, and DEMethod="edgeR", then the function glmWeightedF from the zinbwave package is run; otherwise glmLRT from edgeR.

Note that the argument DEMethod replaces the previous option isCount, to decide on the method of DE.

When 'contrastType' argument implies that the best features should be found via contrasts (i.e. 'contrastType' is 'Pairs' or 'Dendro'), then then 'contrastAdj' determines the type of multiple testing correction to perform. 'PerContrast' does FDR correction for each set of contrasts, and does not guarantee control across all the different contrasts (so probably not the preferred method). 'All' calculates the corrected p-values based on FDR correction of all of the contrasts tested. 'AfterF' controls the FDR based on a hierarchical scheme that only tests the contrasts in those genes where the omnibus F statistic is significant. If the user selects 'AfterF', the user must also supply an option 'p.value' to have any effect, and then only those significant at that p.value level will be returned. Note that currently the correction for 'AfterF' is not guaranteed to control the FDR; improvements will be added in the future.

Note that the default option for topTable is to not filter based on adjusted p-values (p.value = 1) and return only the top 10 most significant (number = 10) – these are options the user can change (these arguments are passed via the ... in getBestFeatures). In particular, it only makes sense to set requireF = TRUE if p.value is meaningful (e.g. 0.1 or 0.05); the default value of p.value = 1 will not result in any effect on the adjusted p-value otherwise.

Value

A data.frame in the same format as topTable, or topTags. The output differs between these two programs, mainly in the naming of columns. Furthermore, if weights are used, an additional column padjFilter is included as the result of running glmWeightedF with default option independentFiltering = TRUE. The following column names are the same between all of the DE methods.

- Feature This is the column called 'ProbeID' by topTable
- IndexInOriginal Gives the index of the feature to the original input dataset, x
- Contrast The contrast that the results corresponds to (if applicable, depends on contrastType argument)
- ContrastName The name of the contrast that the results corresponds to. For dendrogram searches, this will be the node of the tree of the dendrogram. If x is a ClusterExperiment object, this name will make use of the user defined names of the cluster or node in x.
- InternalName Only present if x is a ClusterExperiment object. In this case this column will give the name of the contrast using the internal ids of the clusters and nodes, not the user-defined names. This provides stability in matching the contrast if the user has changed the names since running getBestFeatures
- P.Value The unadjusted p-value (changed from PValue in topTags)
- adj.P.Val The adjusted p-value (changed from FDR or FWER in topTags)

References

Ritchie, ME, Phipson, B, Wu, D, Hu, Y, Law, CW, Shi, W, and Smyth, GK (2015). limma powers differential expression analyses for RNA-sequencing and microarray studies. Nucleic Acids Research 43, e47. http://nar.oxfordjournals.org/content/43/7/e47

Law, CW, Chen, Y, Shi, W, and Smyth, GK (2014). Voom: precision weights unlock linear model analysis tools for RNA-seq read counts. Genome Biology 15, R29. http://genomebiology.com/2014/15/2/R29

Smyth, G. K. (2004). Linear models and empirical Bayes methods for assessing differential expression in microarray experiments. Statistical Applications in Genetics and Molecular Biology, Volume 3, Article 3. http://www.statsci.org/smyth/pubs/ebayes.pdf

See Also

glmLRT glmWeightedF topTable topTags

Examples

```
data(simData)

#create a clustering, for 8 clusters (truth was 4)
cl <- clusterSingle(simData, subsample=FALSE,
sequential=FALSE, mainClusterArgs=list(clusterFunction="pam", clusterArgs=list(k=8)))

#basic F test, return all, even if not significant:
testF <- getBestFeatures(cl, contrastType="F", number=nrow(simData),
DEMethod="limma")

#Do all pairwise, only return significant, try different adjustments:
pairsPerC <- getBestFeatures(cl, contrastType="Pairs", contrastAdj="PerContrast",
p.value=0.05, DEMethod="limma")
pairsAfterF <- getBestFeatures(cl, contrastType="Pairs", contrastAdj="AfterF",
p.value=0.05, DEMethod="limma")
pairsAll <- getBestFeatures(cl, contrastType="Pairs", contrastAdj="All",
p.value=0.05, DEMethod="limma")

#not useful for this silly example, but could look at overlap with Venn
allGenes <- paste("Row", 1:nrow(simData),sep="")
if(require(limma)){
 vennC <- vennCounts(cbind(PerContrast= allGenes %in% pairsPerC$Feature,
 AllJoint=allGenes %in% pairsAll$Feature, FHier=allGenes %in%
 pairsAfterF$Feature))
vennDiagram(vennC, main="FDR Overlap")
}

#Do one cluster against all others
oneAll <- getBestFeatures(cl, contrastType="OneAgainstAll", contrastAdj="All",
p.value=0.05,DEMethod="limma")

#Do dendrogram testing
hcl <- makeDendrogram(cl)
allDendro <- getBestFeatures(hcl, contrastType="Dendro", contrastAdj=c("All"),
number=ncol(simData), p.value=0.05,DEMethod="limma")

# do DE on counts using voom
# compare results to if used simData instead (not on count scale).
testFVoom <- getBestFeatures(simCount, primaryCluster(cl), contrastType="F",
```

```
number=nrow(simData), DEMethod="limma-voom")
plot(testF$P.Value[order(testF$Index)],
testFVoom$P.Value[order(testFVoom$Index)],log="xy")

# do DE on counts using edgeR, compare voom
testFEdge <- getBestFeatures(simCount, primaryCluster(cl), contrastType="F",
n=nrow(simData), DEMethod="edgeR")
plot(testFVoom$P.Value[order(testFVoom$Index)],
testFEdge$P.Value[order(testFEdge$Index)],log="xy")
```

`getClusterIndex` *getClusterIndex*

Description

Finds index of clustering in clusterMatrix slot of object based on descriptions of clusters.

Usage

```
## S4 method for signature 'ClusterExperiment'
getClusterIndex(
  object,
  whichClusters,
  noMatch = c("silentlyRemove", "throwError")
)

## S4 method for signature 'ClusterExperiment'
getSingleClusterIndex(object, whichCluster, passedArgs = NULL, ...)
```

Arguments

`object`	a ClusterExperiment object
`whichClusters`	argument that can be either numeric or character vector indicating the clusterings to be used. See details of `getClusterIndex`.
`noMatch`	how to handle if there is no match to an given value of `whichClusters`. "silentlyRemove" means that no error will be given, and the result will be just those that do match (resulting in a vector of length zero if there are none that match). "throwError" means that the function will stop with an error describing the problem with the match.
`whichCluster`	argument that can be a single numeric or character value indicating the *single* clustering to be used. Giving values that result in more than one clustering will result in an error. See details of `getClusterIndex`.
`passedArgs`	other arguments passed to the function (only used internally)
`...`	Not for user use. Argument allows function `getSingleClusterIndex` to catch the wrong argument (the plural `whichClusters` argument rather than singular `whichCluster`).

Details

The function `getClusterIndex` is largely used internally to parse the argument `whichClusters` which is used as an argument extensively across functions in this package. Note that some functions require the match return a single clustering, in which case those functions use the function `getSingleClusterIndex` with the singular argument `whichCluster` and returns an error if it indicates more than one clustering. Furthermore `getSingleClusterIndex` does not allow for any mismatches (`noMatch="throwError"`. Otherwise the parsing of the two arguments `whichClusters` and `whichCluster` is the same, and is described in what follows.

If `whichClusters` is numeric, then the function just returns the numeric values of `whichClusters`, after checking that they are valid. If any are invalid, they are silently removed if `silentlyRemove=TRUE`. The values will be returned *in the order given*, so this argument can also be used to defined by functions to give an ordering for the clusterings (as relevant).

If `whichClusters` is a character value, then it the function attempts to use the character value to identify the clustering. The value of the argument is first matched against a set of "special" values: "workflow","all","none","primaryCluster","dendro" using the argument `match.arg`, which does partial matching. If whichClusters is a vector of values, only the first value of the vector is matched against these values and if it matches, the remaining values are ignored. If it matches one of these values, then the cluster indices are given as follows.

- "workflow"all clusterings in the current workflow (see `workflowClusters`)
- "all"all clusterings, with the primary clustering put first.
- "none"no clusterings
- "primaryCluster"the primary clustering index as given by `primaryClusterIndex`
- "dendro"the index of the clustering given to create the cluster dendrogram, if it exists

@details If `whichClusters` is a character value, but its first element does not match these predesignated values, then all the values of `whichClusters` are attempted to be matched to the `clusterTypes` of the object. Note that there may be more than one clustering that matches a given type. For any entries that do not match a value in `clusterTypes(object)` are then matched based on the value of `clusterLabels` of the object.

Value

`getClusterIndex` returns a vector of all numeric indices that are indicated by the requested `whichClusters`. Note that there is not a one-to-one match between input values and returned values since there may be more than one value for a given value of `whichClusters` or no value at all.

Examples

```
#load CE object
data(rsecFluidigm)
# get the cluster index from mergeClusters step
getClusterIndex(rsecFluidigm,whichClusters="mergeClusters")
# get the cluster indices from mergeClusters step
getClusterIndex(rsecFluidigm,whichClusters="clusterMany")
```

`getClusterManyParams,ClusterExperiment-method`
Get parameter values of clusterMany clusters

Description

Takes an input a ClusterExperiment object and returns the parameter values used in creating the clusters that were created by 'clusterMany'

Usage

```
## S4 method for signature 'ClusterExperiment'
getClusterManyParams(
  x,
  whichClusters = "clusterMany",
  searchAll = FALSE,
  simplify = TRUE
)
```

Arguments

`x`	a ClusterExperiment object that contains clusterings from running `clusterMany`.
`whichClusters`	argument that can be either numeric or character vector indicating the clusterings to be used. See details of `getClusterIndex`.
`searchAll`	logical, indicating whether all clusterings with `clusterMany` label should be allowed (i.e. including those from previous ones with labels like `clusterMany.1`), or only limited to those in most recent workflow (default).
`simplify`	logical. Whether to simplify the output so as to remove features that do not change across the clusterings.

Details

The method simply parses the `clusterLabels` of the indicated clusterings, relying on the specific format used by `clusterMany` to create labels. The function will only allow the parsing to be performed on those clusterings with a 'clusterMany' clusterType. If the user has manipulated the clusterLabels manually or manually identified the clusterType of a clustering as 'clusterMany', this function may create unexpected results or errors. Similarly, it cannot be used on 'clusterMany' results from an old iteration (e.g. that have type 'clusterMany.1')

Specifically, it splits the label of each clustering by the character ",", as indicating the different parameters; this should return a value of form "ABC=123". The function then pulls out the numeric value ('123') and associates that value as the value of the parameter ('ABC')

Value

Returns a data.frame where the column names are the parameter names, and the entries are the values of the parameter for the indicated clustering. The column 'clusteringIndex' identifies the index of the clustering in the full set of clusterings of the given ClusterExperiment object.

Examples

```
data(simData)

cl <- clusterMany(simData, nReducedDims=c(5, 10), reduceMethod="PCA",
     clusterFunction="pam", ks=2:4, findBestK=c(TRUE,FALSE),
     makeMissingDiss=TRUE, removeSil=c(TRUE,FALSE))
getClusterManyParams(cl)
```

---

getReducedData,ClusterExperiment-method

Return matrix from ClusterExperiment with reduced dimensions

Description

Returns a matrix of data from a `ClusterExperiment` object based on the choices of dimensionality reduction given by the user.

Functions for calculating and manipulating either filtering statistics, stored in rowData, or the dimensionality reduction results, stored in reducedDims.

Usage

```
## S4 method for signature 'ClusterExperiment'
getReducedData(
  object,
  reduceMethod,
  filterIgnoresUnassigned,
  nDims = defaultNDims(object, reduceMethod),
  whichCluster = "primary",
  whichAssay = 1,
  returnValue = c("object", "list"),
  reducedDimName
)

## S4 method for signature 'SingleCellExperiment'
defaultNDims(object, reduceMethod, typeToShow)

## S4 method for signature 'matrixOrHDF5'
defaultNDims(object, ...)

## S4 method for signature 'SummarizedExperiment'
makeFilterStats(
  object,
  filterStats = listBuiltInFilterStats(),
  transFun = NULL,
  isCount = FALSE,
  filterNames = NULL,
  whichAssay = 1
)

## S4 method for signature 'matrixOrHDF5'
makeFilterStats(object, ...)
```

```
## S4 method for signature 'ClusterExperiment'
makeFilterStats(
  object,
  whichClusterIgnoreUnassigned = NULL,
  filterStats = listBuiltInFilterStats(),
  ...
)

listBuiltInFilterStats()

## S4 method for signature 'SummarizedExperiment'
filterData(
  object,
  filterStats,
  cutoff,
  percentile,
  absolute = FALSE,
  keepLarge = TRUE,
  whichAssay = 1
)

## S4 method for signature 'SummarizedExperiment'
filterNames(object)

## S4 method for signature 'SingleCellExperiment'
makeReducedDims(
  object,
  reducedDims = "PCA",
  maxDims = 500,
  transFun = NULL,
  isCount = FALSE,
  whichAssay = 1
)

## S4 method for signature 'matrixOrHDF5'
makeReducedDims(object, ...)

## S4 method for signature 'SummarizedExperiment'
makeReducedDims(object, ...)

## S4 method for signature 'ClusterExperiment'
makeReducedDims(object, ...)

listBuiltInReducedDims()
```

Arguments

object	For makeReducedDims,makeFilterStats, defaultNDims either matrix-like, SingleCellExperimer or ClusterExperiment object. For getReducedData only a ClusterExperiment object allowed.
reduceMethod	character. A method (or methods) for reducing the size of the data, either

by filtering the rows (genes) or by a dimensionality reduction method. Must either be 1) must match the name of a built-in method, in which case if it is not already existing in the object will be passed to `makeFilterStats` or `link{makeReducedDims}`, or 2) must match a stored filtering statistic or dimensionality reduction in the object

`filterIgnoresUnassigned`
: logical. Whether filtering statistics should ignore the unassigned samples within the clustering. Only relevant if 'reduceMethod' matches one of built-in filtering statistics in `listBuiltInFilterStats()`, in which case the clustering identified in `whichCluster` is passed to `makeFilterStats` and the unassigned samples are excluded in calculating the statistic. See `makeFilterStats` for more details.

`nDims`
: The number of dimensions to keep from `reduceMethod`. If missing calls `defaultNDims`.

`whichCluster`
: argument that can be a single numeric or character value indicating the *single* clustering to be used. Giving values that result in more than one clustering will result in an error. See details of `getClusterIndex`.

`whichAssay`
: numeric or character specifying which assay to use. See `assay` for details.

`returnValue`
: The format of output. Users will generally want to keep the default (see details)

`reducedDimName`
: The name given to the reducedDims slot storing result (if `returnValue="object"`). If missing, the function will create a default name: if `reduceMethod` is a dimensionality reduction, then `reduceMethod` will be given as the name; if a filtering statistic, "filteredBy_" followed by `reduceMethod`.

`typeToShow`
: character (optional). If given, should be one of "filterStats" or "reducedDims" to indicate of the values in the reduceMethod vector, only show those corresponding to "filterStats" or "reducedDims" options.

`...`
: Values passed on the the 'SingleCellExperiment' method.

`filterStats`
: character vector of statistics to calculate. Must be one of the character values given by `listBuildInFilterStats()`.

`transFun`
: a transformation function to be applied to the data. If the transformation applied to the data creates an error or NA values, then the function will throw an error. If object is of class `ClusterExperiment`, the stored transformation will be used and giving this parameter will result in an error.

`isCount`
: if `transFun=NULL`, then `isCount=TRUE` will determine the transformation as defined by `function(x){log2(x+1)}`, and `isCount=FALSE` will give a transformation function `function(x){x}`. Ignored if `transFun=NULL`. If object is of class `ClusterExperiment`, the stored transformation will be used and giving this parameter will result in an error.

`filterNames`
: if given, defines the names that will be assigned to the filtering statistics in the `rowData` of the object. If missing, will be just the value of `filterStats` argument

`whichClusterIgnoreUnassigned`
: indicates clustering that should be used to filter out unassigned samples from the calculations. If `NULL` no filtering of samples will be done. See details for more information.

`cutoff`
: numeric. A value at which to filter the rows (genes) for the test statistic

`percentile`
: numeric. Either a number between 0,1 indicating what percentage of the rows (genes) to keep or an integer value indicated the number of rows (genes) to keep

`absolute`
: whether to take the absolute value of the filter statistic

keepLarge	logical whether to keep rows (genes) with large values of the test statistic or small values of the test statistic.
reducedDims	a vector of character values indicating the methods of dimensionality reduction to be performed. Currently only "PCA" is implemented.
maxDims	Numeric vector of integer giving the number of PC dimensions to calculate. maxDims can also take values between (0,1) to indicate keeping the number of dimensions necessary to account for that proportion of the variance. maxDims should be of same length as reducedDims, indicating the number of dimensions to keep for each method (if maxDims is of length 1, the same number of dimensions will be used for each).

Details

getReducedData determines the matrix of values that can be used for computation based on the user's choice of dimensionality methods. The methods can be either of the filtering kind or the more general dimensionality reduction. The function will first look at any stored ReducedDims or filtering statistics already present in the data, and if missing, will assume that reduceMethod is one of the built-in method provided by the package and calculate the necessary. Note that if reduceMethod is a filtering statistic, in addition to filtering the features, the function will also perform the stored transformation of the data.

Note that this is used internally by functions, but is mainly only of interest for the user if they want to have the filtered, transformed data available as a matrix for continual use.

If returnValue="object", then the output is a single, updated ClusterExperiment object with the reduced data matrix stored as an element of the list in reducedDims slot (with name given by reducedDimName if given). If "list", then a list with one element that is the object and the other that is the reduced data matrix. Either way, the object returned in the list will be updated to contain with the filtering statistics or the dimensionality reduction. The only difference is that if "list", the reduced dimension matrix is NOT saved in the object (and so only really makes a difference if the reduceMethod argument is a filtering method). The option "list" is mainly for internal use, where we do not want to continually save subseted datasets.

If nDims is missing, it will be given a default value depending on the value of reduceMethod. See defaultNDims for details.

If filterIgnoresUnassigned is missing, then it is set to TRUE *unless*: reduceMethod matches a stored filtering statistic in rowData AND does not match a built-in filtering method provided by the package.

For a reduceMethod that corresponds to a filtering statistics the current default is 1000 (or the length of the number of features, if less). For a dimensionality reduction saved in the reducedDims slot the default is 50 or the maximum number of dimensions if less than 50.

reduceMethod will first be checked to see if it corresponds with an existing saved filtering statistic or a dimensionality reduction to determine which of these two types it is. If it does not match either, then it will be checked against the built in functions provided by the package. @examples se<-SingleCellExperiment(matrix(rnorm(5000*100),nrow=5000,ncol=100)) defaultNDims(se,"PCA") defaultNDims(se,"mad")

whichClusterIgnoreUnassigned is only an option when applied to a ClusterExperiment classs and indicates that the filtering statistics should be calculated based on samples that are unassigned by the designated clustering. The name given to the filter in this case is of the form <filterStats>_<clusterLabel>, i.e. the clustering label of the clustering is appended to the standard name for the filtering statistic.

Note that filterData returns a SingleCellExperiment object. To get the actual data out use either assay or transformData if transformed data is desired.

The PCA method uses either prcomp from the stats package or svds from the RSpectra package to perform PCA. Both are called on t(assay(x)) with center=TRUE and scale=TRUE (i.e. the feature are centered and scaled), so that it is performing PCA on the correlation matrix of the features.

Note that this function does not check if such a reduceDim value already exists, and will recalculate (and overwrite) if it does.

Value

If returnValue="object", a ClusterExperiment object.

If returnValue="list" a list with elements:

- objectUpdateobject, potentially updated if had to calculate dimensionality reduction or filtering statistic
- dataMatrixthe reduced dimensional matrix with the samples in columns, features in rows

defaultNDims returns a numeric vector giving the default dimensions the methods in clusterExperiment will use for reducing the size of the data. If typeToShow is missing, the resulting vector will be equal to the length of reduceMethod. Otherwise, it will be a vector with all the unique valid default values for the typeToShow (note that different dimensionality reduction methods can have different maximal dimensions, so the result may not be of length one in this case).

makeFilterStats returns a SummarizedExperiment object with the requested filtering statistics will be added to the DataFrame in the rowData slot and given names corresponding to the filterStats values. Warning: the function will overwrite existing columns in rowData with the same name. Columns in the rowData slot with different names should not be affected.

filterData returns a SingleCellExperiment object with the rows (genes) removed based on filters

filterNames returns a vector of the columns of the rowData that are considered valid filtering statistics. Currently any numeric column in rowData is a valid filtering statistic.

makeReducedDims returns a SingleCellExperiment containing the calculated dimensionality reduction in the reduceDims with names corresponding to the name given in reducedDims.

See Also

makeFilterStats,makeReducedDims, filterData, reducedDim

Examples

```
data(simData)
listBuiltInFilterStats()
scf<-makeFilterStats(simData,filterStats=c("var","mad"))
scf
scfFiltered<-filterData(scf,filterStats="mad",percentile=10)
scfFiltered
assay(scfFiltered)[1:10,1:10]
data(simData)
listBuiltInReducedDims()
scf<-makeReducedDims(simData, reducedDims="PCA", maxDims=3)
scf
```

`internalFunctionCheck` *Class ClusterFunction*

Description

`ClusterFunction` is a class for holding functions that can be used for clustering in the clustering algorithms in this package.

The constructor `ClusterFunction` creates an object of the class `ClusterFunction`.

Usage

```
internalFunctionCheck(clusterFUN, inputType, algorithmType, outputType)

ClusterFunction(clusterFUN, ...)

## S4 method for signature '`function`'
ClusterFunction(
  clusterFUN,
  inputType,
  outputType,
  algorithmType,
  inputClassifyType = NA_character_,
  requiredArgs = NA_character_,
  classifyFUN = NULL,
  checkFunctions = TRUE
)
```

Arguments

`clusterFUN`	function passed to slot `clusterFUN`.
`inputType`	character for slot `inputType`
`algorithmType`	character for slot `inputType`
`outputType`	character for slot `outputType`
`...`	arguments passed to different methods of `ClusterFunction`
`inputClassifyType`	character for slot `inputClassifyType`
`requiredArgs`	character for slot `requiredArgs`
`classifyFUN`	function for slot `classifyFUN`
`checkFunctions`	logical for whether to check the input functions with `internalFunctionsCheck`

Details

`internalFunctionCheck` is the function that is called by the validity check of the `ClusterFunction` constructor (if `checkFunctions=TRUE`). It is available as an S3 function for the user to be able to test their functions and debug them, which is difficult to do with a S4 validity function.

clusterFUN: The following arguments are required to be accepted for `clusterFUN` – higher-level code may pass these arguments (but the function can ignore them or just have be handled with a ...)

- "inputMatrix"will be the matrix of data
- "inputType"one of "X", "diss", or "cat". If "X", then `inputMatrix` is assumed to be nfeatures x nsamples (like assay(CEObj) would give). If "cat" then nfeatures x nsamples, but all entries should be categorical levels, encoded by positive integers, with -1/-2 types of NA (like a clusterMatrix slot, but with dimensions switched). If "diss", then `inputMatrix` should be a nxn dissimilarity matrix.
- "checkArgs"logical argument. If `checkArgs=TRUE`, the `clusterFUN` should check if the arguments passed in `...` are valid and return an error if not; otherwise, no error will be given, but the check should be done and only valid arguments in `...` passed along. This is necessary for the function to work with `clusterMany` which passes all arguments to all functions without checking.
- "cluster.only"logical argument. If `cluster.only=TRUE`, then `clusterFUN` should return only the vector of cluster assignments (or list if `outputType="list"`). If `cluster.only=FALSE` then the `clusterFUN` should return a named list where one of the elements entitled `clustering` contains the vector described above (no list allowed!); anything else needed by the `classifyFUN` to classify new data should be contained in the output list as well. `cluster.only` is set internally depending on whether `classifyFUN` will be later used by subsampling or only for clustering the final product.
- "..."Any additional arguments specific to the algorithm used by `clusterFUN` should be passed via `...` and NOT passed via arguments to `clusterFUN`
- "Other required arguments"`clusterFUN` must also accept arguments required for its `algorithmType` (see Details below).

classifyFUN: The following arguments are required to be accepted for `classifyFUN` (if not NULL)

- inputMatrixthe *new* data that will be classified into the clusters
- inputTypethe inputType of the new data (see above)
- clusterResultthe result of running `clusterFUN` on the training data, when `cluster.only=FALSE`. Whatever is returned by `clusterFUN` is assumed to be sufficient for this function to classify new objects (e.g. could return the centroids of the clustering, if clustering based on nearest centroid).

algorithmType: Type "01" is for clustering functions that expect as an input a dissimilarity matrix that takes on 0-1 values (e.g. from subclustering) with 1 indicating more dissimilarity between samples. "01" algorithm types must also have `inputType` equal to `"diss"`. It is also generally expected that "01" algorithms use the 0-1 nature of the input to set criteria as to where to find clusters. "01" functions must take as an argument `alpha` between 0 and 1 to determine the clusters, where larger values of `alpha` require less similarity between samples in the same cluster. "K" is for clustering functions that require an argument `k` (the number of clusters), but arbitrary `inputType`. On the other hand, "K" algorithms are assumed to need a predetermined 'k' and are also assumed to cluster all samples to a cluster. If not, the post-processing steps in `mainClustering` such as `findBestK` and `removeSil` may not operate correctly since they rely on silhouette distances.

Value

Returns a logical value of TRUE if there are no problems. If there is a problem, returns a character string describing the problem encountered.

A `ClusterFunction` object.

Slots

`clusterFUN` a function defining the clustering function. See details for required arguments.

`inputType` a character vector defining what type(s) of input `clusterFUN` takes. Must consist of values "diss","X", or "cat" indicating the set of input values that the algorithm can handle (see details below).

`algorithmType` a character defining what type of clustering algorithm `clusterFUN` is. Must be one of either "01" or "K". `clusterFUN` must take the corresponding required arguments for its type (see details below).

`classifyFUN` a function that has takes as input new data and the output of `clusterFUN` (where the output is from when `cluster.only=FALSE`) and results in cluster assignments of the new data. Used in subsampling clustering. Note that the function should assume that the data given to the `inputMatrix` argument is not the same samples that were input to the ClusterFunction (but does assume that it is the same number of features/columns). If slot `classifyFUN` is given value `NULL` then subsampling type can only be `"InSample"`, see `subsampleClustering`.

`inputClassifyType` the input type for the classification function (if not NULL); like `inputType`, must be a vector containing "diss","X", or "cat"

`outputType` the type of output given by `clusterFUN`. Must either be "vector" or "list". If "vector" then the output should be a vector of length equal to the number of observations with integer-valued elements identifying them to different clusters; the vector assignments should be in the same order as the original input of the data. Samples that are not assigned to any cluster should be given a '-1' value. If "list", then it must be a list equal to the length of the number of clusters, and the elements of the list contain the indices of the samples in that cluster. Any indices not in any of the list elements are assumed to be -1. The main advantage of "list" is that it can preserve the order of the clusters if the `clusterFUN` desires to do so. In which case the `orderBy` argument of `mainClustering` can preserve this ordering (default is to order by size).

`requiredArgs` Any additional required arguments for `clusterFUN` (beyond those required of all `clusterFUN`, described in details). Will be used in checking that user provided necessary arguments.

`checkFunctions` logical. If TRUE, the validity check of the `ClusterFunction` object will check the `clusterFUN` with simple toy data using the function `internalFunctionCheck`.

Examples

```
#Use internalFunctionCheck to check possible function
goodFUN<-function(inputMatrix,k,cluster.only,...){
cluster::pam(x=t(inputMatrix),k=k,cluster.only=cluster.only)
}
#passes internal check
internalFunctionCheck(goodFUN,inputType=c("X","diss"),
   algorithmType="K",outputType="vector")
myCF<-ClusterFunction(clusterFUN=goodFUN, inputType="X",
   algorithmType="K", outputType="vector")
#doesn't work, because haven't made results return vector when cluster.only=TRUE
badFUN<-function(inputMatrix,k,cluster.only,...){cluster::pam(x=inputMatrix,k=k)}
internalFunctionCheck(badFUN,inputType=c("X","diss"),
   algorithmType="K",outputType="vector")
```

listBuiltInFunctions *Built in ClusterFunction options*

Description

Documents the built-in clustering options that are available in the clusterExperiment package.

Usage

```
listBuiltInFunctions()

## S4 method for signature 'character'
getBuiltInFunction(object)

listBuiltInTypeK()

listBuiltInType01()
```

Arguments

object	name of built in function.

Details

listBuiltInFunctions will return the character names of the built-in clustering functions available.

listBuiltInTypeK returns the names of the built-in functions that have type 'K'

listBuiltInType01 returns the names of the built-in functions that have type '01'

getBuiltInFunction will return the ClusterFunction object of a character value that corresponds to a built-in function.

algorithmType and inputType will return the algorithmType and inputType of the built-in clusterFunction corresponding to the character value.

Built-in clustering methods: The built-in clustering methods, the names of which can be accessed by listBuiltInFunctions() are the following:

- "pam"Based on pam in cluster package. Arguments to that function can be passed via clusterArgs. Input can be either "x" or "diss"; algorithm type is "K"
- "clara"Based on clara in cluster package. Arguments to that function can be passed via clusterArgs. Note that we have changed the default arguments of that function to match the recommendations in the documentation of clara (numerous functions are set to less than optimal settings for back-compatiability). Specifically, the following defaults are implemented samples=50, keep.data=FALSE, mediods.x=FALSE,rngR=TRUE, pamLike=TRUE, correct.d=TRUE. Input is "X"; algorithm type is "K".
- "kmeans"Based on kmeans in stats package. Arguments to that function can be passed via clusterArgs except for centers which is reencoded here to be the argument 'k' Input is "X"; algorithm type is "K"
- "mbkmeans"mbkmeans runs mini-batch kmeans, a more computationally efficient version of kmeans.

- "hierarchical01"`hclust` in `stats` package is used to build hiearchical clustering. Arguments to that function can be passed via `clusterArgs`. The `hierarchical01` cuts the hiearchical tree based on the parameter `alpha`. It does not use the `cutree` function, but instead transversing down the tree until getting a block of samples with whose summary of the values is greater than or equal to 1-alpha. Arguments that can be passed to 'hierarchical01' are 'evalClusterMethod' which determines how to summarize the samples' values of D[samples,samples] for comparison to 1-alpha: "maximum" (default) takes the minimum of D[samples,samples] and requires it to be less than or equal to 1-alpha; "average" requires that each row mean of D[samples,samples] be less than or equal to 1-alpha. Additional arguments of hclust can also be passed via clusterArgs to control the hierarchical clustering of D. Input is `"diss"`; algorithm type is "01"
- "hierarchicalK"`hclust` in `stats` package is used to build hiearchical clustering and `cutree` is used to cut the tree into `k` clusters. Input is `"diss"`; algorithm type is "K"
- "tight"Based on the algorithm in Tsang and Wong, specifically their method of picking clusters from a co-occurance matrix after subsampling. The clustering encoded here is not the entire tight clustering algorithm, only that single piece that identifies clusters from the co-occurance matrix. Arguments for the tight method are 'minSize.core' (default=2), which sets the minimimum number of samples that form a core cluster. Input is `"diss"`; algorithm type is "01"
- "spectral"`specc` in `kernlab` package is used to perform spectral clustering. Note that spectral clustering can produce errors if the number of clusters (K) is not sufficiently smaller than the number of samples (N). K < N is not always sufficient. Input is `"X"`; algorithm type is "K".

Value

`listBuiltInFunctions` returns a character vector of all the built-in cluster functions' names.

`getBuiltInFunction` returns the `ClusterFunction` object that corresponds to the character name of a function

`listBuiltInTypeK` returns a character vector of the names of built-in cluster functions that are of type "K"

`listBuiltInType01` returns a character vector of the names of built-in cluster functions that are of type "01"

See Also

`ClusterFunction`, `algorithmType`, `inputType`

Examples

```
listBuiltInFunctions()
algorithmType(c("kmeans","pam","hierarchical01"))
inputType(c("kmeans","pam","hierarchical01"))
listBuiltInTypeK()
listBuiltInType01()
```

mainClustering *Cluster distance matrix from subsampling*

Description

Given input data, this function will try to find the clusters based on the given ClusterFunction object.

Usage

```
## S4 method for signature 'character'
mainClustering(clusterFunction, ...)

## S4 method for signature 'ClusterFunction'
mainClustering(
  clusterFunction,
  inputMatrix,
  inputType,
  clusterArgs = NULL,
  minSize = 1,
  orderBy = c("size", "best"),
  format = c("vector", "list"),
  returnData = FALSE,
  warnings = TRUE,
  ...
)

## S4 method for signature 'ClusterFunction'
getPostProcessingArgs(clusterFunction)
```

Arguments

clusterFunction	a ClusterFunction object that defines the clustering routine. See ClusterFunction for required format of user-defined clustering routines. User can also give a character value to the argument clusterFunction to indicate the use of clustering routines provided in package. Type listBuiltInFunctions at command prompt to see the built-in clustering routines. If clusterFunction is missing, the default is set to "pam".
...	arguments passed to the post-processing steps of the clustering. The available post-processing arguments for a ClusterFunction object depend on it's algorithm type and can be found by calling getPostProcessingArgs. See details below for documentation.
inputMatrix	numerical matrix on which to run the clustering or a SummarizedExperiment, SingleCellExperiment, or ClusterExperiment object.
inputType	a character vector defining what type of input is given in the inputMatrix argument. Must consist of values "diss","X", or "cat" (see details). "X" and "cat" should be indicate matrices with features in the row and samples in the column; "cat" corresponds to the features being numerical integers corresponding to categories, while "X" are continuous valued features. "diss" corresponds to an inputMatrix that is a NxN dissimilarity matrix. "cat" is largely used internally for clustering of sets of clusterings.

clusterArgs	arguments to be passed directly to the clusterFUN slot of the ClusterFunction object
minSize	the minimum number of samples in a cluster. Clusters found below this size will be discarded and samples in the cluster will be given a cluster assignment of "-1" to indicate that they were not clustered.
orderBy	how to order the cluster (either by size or by maximum alpha value). If orderBy="size" the numbering of the clusters are reordered by the size of the cluster, instead of by the internal ordering of the clusterFUN defined in the ClusterFunction object (an internal ordering is only possible if slot outputType of the ClusterFunction is "list").
format	whether to return a list of indices in a cluster or a vector of clustering assignments. List is mainly for compatibility with sequential part.
returnData	logical as to whether to return the diss or x matrix in the output. If FALSE only the clustering vector is returned.
warnings	logical as to whether should give warning if arguments given that don't match clustering choices given. Otherwise, inapplicable arguments will be ignored without warning.

Details

mainClustering is not meant to be called by the user. It is only an exported function so as to be able to clearly document the arguments for mainClustering which can be passed via the argument mainClusterArgs in functions like clusterSingle and clusterMany.

Post-processing Arguments: For post-processing the clustering, currently only type 'K' algorithms have a defined post-processing. Specifically

- "findBestK"logical, whether should find best K based on average silhouette width (only used if clusterFunction of type "K").
- "kRange"vector of integers to try for k values if findBestK=TRUE. If k is given in clusterArgs, then default is k-2 to k+20, subject to those values being greater than 2; if not the default is 2:20. Note that default values depend on the input k, so running for different choices of k and findBestK=TRUE can give different answers unless kRange is set to be the same.
- "removeSil"logical as to whether remove the assignment of a sample to a cluster when the sample's silhouette value is less than silCutoff
- "silCutoff"Cutoff on the minimum silhouette width to be included in cluster (only used if removeSil=TRUE).

Value

If returnData=FALSE, mainClustering returns a vector of cluster assignments (if format="vector") or a list of indices for each cluster (if format="list"). Clusters less than minSize are removed. If returnData=TRUE, then mainClustering returns a list

- resultsThe clusterings of each sample.
- inputMatrixThe input matrix given to argument inputMatrix. Useful if input is result of subsampling, in which case input is the set of clusterings found over subsampling.

Examples

```
data(simData)
cl1<-mainClustering(inputMatrix=simData, inputType="X",
    clusterFunction="pam",clusterArgs=list(k=3))
#supply a dissimilarity, use algorithm type "01"
diss<-as.matrix(dist(t(simData),method="manhattan"))
cl2<-mainClustering(diss, inputType="diss", clusterFunction="hierarchical01",
    clusterArgs=list(alpha=.1))
cl3<-mainClustering(inputMatrix=diss, inputType="diss", clusterFunction="pam",
    clusterArgs=list(k=3))

# run hierarchical method for finding blocks, with method of evaluating
# coherence of block set to evalClusterMethod="average", and the hierarchical
# clustering using single linkage:
# (clustering function requires type 'diss'),
clustSubHier <- mainClustering(diss, inputType="diss",
    clusterFunction="hierarchical01", minSize=5,
    clusterArgs=list(alpha=0.1,evalClusterMethod="average", method="single"))

#post-process results of pam -- must pass diss for silhouette calculation
clustSubPamK <- mainClustering(simData, inputType="X", clusterFunction="pam",
    silCutoff=0, minSize=5, diss=diss, removeSil=TRUE, clusterArgs=list(k=3))
clustSubPamBestK <- mainClustering(simData, inputType="X", clusterFunction="pam", silCutoff=0,
    minSize=5, diss=diss, removeSil=TRUE, findBestK=TRUE, kRange=2:10)

# note that passing the wrong arguments for an algorithm results in warnings
# (which can be turned off with warnings=FALSE)
clustSubTight_test <- mainClustering(diss, inputType="diss",
   clusterFunction="tight",
   clusterArgs=list(alpha=0.1), minSize=5, removeSil=TRUE)
clustSubTight_test2 <- mainClustering(diss, inputType="diss",
   clusterFunction="tight",
   clusterArgs=list(alpha=0.1,evalClusterMethod="average"))
```

---

makeConsensus *Find sets of samples that stay together across clusterings*

Description

Find sets of samples that stay together across clusterings in order to define a new clustering vector.

Usage

```
## S4 method for signature 'matrix'
makeConsensus(
  x,
  proportion,
  clusterFunction = "hierarchical01",
  minSize = 5,
  propUnassigned = 0.5,
  whenUnassign = c("before", "after"),
  clusterArgs = NULL
)
```

```
## S4 method for signature 'ClusterExperiment'
makeConsensus(
  x,
  whichClusters,
  eraseOld = FALSE,
  clusterLabel = "makeConsensus",
  ...
)
```

Arguments

x	a matrix with samples on the rows and different clusterings on the columns or ClusterExperiment object.
proportion	The proportion of times that two sets of samples should be together in order to be grouped into a cluster (if <1, passed to mainClustering via alpha = 1 - proportion)
clusterFunction	the clustering function to use (passed to mainClustering); currently must be of type '01' and accept as input matrices of type "cat" (see details of ?ClusterFunction).
minSize	minimum size required for a set of samples to be considered in a cluster because of shared clustering, passed to mainClustering
propUnassigned	samples with greater than this proportion of assignments equal to '-1' are assigned a '-1' cluster value as a last step (only if proportion < 1)
whenUnassign	(provided for back compatibility with previous versions). Must be one of "before" or "after", indicating at what point are samples with a proportion of assignments of -1 greater than propUnassigned forced to have a '-1' value. If "before", then these samples are removed and not used for clustering. If "after", these samples are included in the clustering step, but then the cluster values they receive are assigned a '-1. These choices may result in different clusterings, because if these samples are included in the clustering (i.e. whenUnassign="after", then these samples may affect the cluster assignments of other samples. The default is currently "before", but previous to version 2.5.4, there was no such option and the code internally set to "after", so for reproducibility with older results, users may need to set this option.
clusterArgs	list of arguments to be passed to the call to mainClustering that is used to cluster the proportion overlap between samples.
whichClusters	argument that can be either numeric or character vector indicating the clusterings to be used. See details of getClusterIndex.
eraseOld	logical. Only relevant if input x is of class ClusterExperiment. If TRUE, will erase existing workflow results (clusterMany as well as mergeClusters and makeConsensus). If FALSE, existing workflow results will have "_i" added to the clusterTypes value, where i is one more than the largest such existing workflow clusterTypes.
clusterLabel	a string used to describe the type of clustering. By default it is equal to "makeConsensus", to indicate that this clustering is the result of a call to makeConsensus. However, a more informative label can be set (see vignette).
...	arguments to be passed on to the method for signature matrix,missing.

Details

This function was previously called combineMany (versions <= 2.0.0). combineMany is still available, but is considered defunct and users should update their code accordingly.

The function tries to find a consensus cluster across many different clusterings of the same samples. It does so by creating a nSamples x nSamples matrix of the percentage of co-occurance of each sample and then calling mainClustering to cluster the co-occurance matrix. The function assumes that '-1' labels indicate clusters that are not assigned to a cluster. Co-occurance with the unassigned cluster is treated differently than other clusters. The percent co-occurance is taken only with respect to those clusterings where both samples were assigned. Then samples with more than propUnassigned values that are '-1' across all of the clusterings are assigned a '-1' regardless of their cluster assignment.

The method calls mainClustering on the proportion matrix with clusterFunction as the 01 clustering algorithm, alpha=1-proportion, minSize=minSize, and evalClusterMethod=c("average"). See help of mainClustering for more details.

Value

If x is a matrix, a list with values

- clustering vector of cluster assignments, with "-1" implying unassigned
- percentageShared a nSample x nSample matrix of the percent co-occurance across clusters used to find the final clusters. Percentage is out of those not '-1'
- noUnassignedCorrection a vector of cluster assignments before samples were converted to '-1' because had >propUnassigned '-1' values (i.e. the direct output of the mainClustering output.)

If x is a ClusterExperiment, a ClusterExperiment object, with an added clustering of clusterTypes equal to makeConsensus and the percentageShared matrix stored in the coClustering slot.

Examples

```
data(simData)

cl <- clusterMany(simData,nReducedDims=c(5,10,50),  reduceMethod="PCA",
clusterFunction="pam", ks=2:4, findBestK=c(FALSE), removeSil=TRUE,
makeMissingDiss=TRUE, subsample=FALSE)

#make names shorter for plotting
clMat <- clusterMatrix(cl)
colnames(clMat) <- gsub("TRUE", "T", colnames(clMat))
colnames(clMat) <- gsub("FALSE", "F", colnames(clMat))
colnames(clMat) <- gsub("k=NA,", "", colnames(clMat))

#require 100% agreement -- very strict
clCommon100 <- makeConsensus(clMat, proportion=1, minSize=10)

#require 70% agreement based on clustering of overlap
clCommon70 <- makeConsensus(clMat, proportion=0.7, minSize=10)

oldpar <- par(no.readonly = TRUE)
par(mar=c(1.1, 12.1, 1.1, 1.1))
plotClusters(cbind("70%Similarity"=clCommon70, clMat,
"100%Similarity"=clCommon100), axisLine=-2)
```

```
#method for ClusterExperiment object
clCommon <- makeConsensus(cl, whichClusters="workflow", proportion=0.7,
minSize=10)
plotClusters(clCommon)
par(oldpar)
```

---

makeDendrogram *Make hierarchy of set of clusters*

Description

Makes a dendrogram of a set of clusters based on hclust on the medoids of the cluster.

Usage

```
## S4 method for signature 'ClusterExperiment'
makeDendrogram(
  x,
  whichCluster = "primaryCluster",
  reduceMethod = "mad",
  nDims = defaultNDims(x, reduceMethod),
  filterIgnoresUnassigned = TRUE,
  unassignedSamples = c("outgroup", "cluster"),
  whichAssay = 1,
  ...
)

## S4 method for signature 'dist'
makeDendrogram(
  x,
  cluster,
  unassignedSamples = c("outgroup", "cluster", "remove"),
  calculateSample = TRUE,
  ...
)

## S4 method for signature 'matrixOrHDF5'
makeDendrogram(
  x,
  cluster,
  unassignedSamples = c("outgroup", "cluster", "remove"),
  calculateSample = TRUE,
  ...
)
```

Arguments

x	data to define the medoids from. Matrix and ClusterExperiment supported.

`whichCluster` argument that can be a single numeric or character value indicating the *single* clustering to be used. Giving values that result in more than one clustering will result in an error. See details of `getClusterIndex`.

`reduceMethod` character A character identifying what type of dimensionality reduction to perform before clustering. Can be either a value stored in either of reducedDims or filterStats slot or a built-in diminsionality reduction/filtering. The option "coCluster" will use the co-Clustering matrix stored in the 'coClustering' slot of the `ClusterExperiment` object

`nDims` The number of dimensions to keep from `reduceMethod`. If missing calls `defaultNDims`.

`filterIgnoresUnassigned` logical. Whether filtering statistics should ignore the unassigned samples within the clustering. Only relevant if 'reduceMethod' matches one of built-in filtering statistics in `listBuiltInFilterStats()`, in which case the clustering identified in `whichCluster` is passed to `makeFilterStats` and the unassigned samples are excluded in calculating the statistic. See `makeFilterStats` for more details.

`unassignedSamples` how to handle unassigned samples("-1") ; only relevant for sample clustering. See details.

`whichAssay` numeric or character specifying which assay to use. See `assay` for details.

`...` for makeDendrogram, if signature `matrix`, arguments passed to hclust; if signature `ClusterExperiment` passed to the method for signature `matrix`. For plotDendrogram, passed to `plot.dendrogram`.

`cluster` A numeric vector with cluster assignments. If x is a ClusterExperiment object, cluster is automatically the primaryCluster(x). "-1" indicates the sample was not assigned to a cluster.

`calculateSample` only relevant for `matrix` or `dist` version of function. Indicates whether to calculate the sample dendrogram.

Details

The function returns two dendrograms (as a list if x is a matrix or in the appropriate slots if x is ClusterExperiment). The cluster dendrogram is created by applying `hclust` to the medoids of each cluster. In the sample dendrogram the clusters are again clustered, but now the samples are also part of the resulting dendrogram. This is done by giving each sample the value of the medoid of its cluster.

The argument `unassignedSamples` governs what is done with unassigned samples (defined by a -1 cluster value). If unassigned=="cluster", then the dendrogram is created by hclust of the expanded medoid data plus the original unclustered observations. If `unassignedSamples` is "outgroup", then all unassigned samples are put as an outgroup. If the `x` object is a matrix, then `unassignedSamples` can also be "remove", to indicate that samples with "-1" should be discarded. This is not a permitted option, however, when `x` is a `ClusterExperiment` object, because it would return a dendrogram with less samples than `NCOL(x)`, which is not permitted for the `@dendro_samples` slot.

If any merge information is stored in the input object, it will be erased by a call to makeDendrogram.

Value

If x is a matrix, a list with two dendrograms, one in which the leaves are clusters and one in which the leaves are samples. If x is a ClusterExperiment object, the dendrograms are saved in the appropriate slots.

See Also

makeFilterStats, makeReducedDims

Examples

```
data(simData)

#create a clustering, for 8 clusters (truth was 3)
cl <- clusterSingle(simData, subsample=FALSE,
sequential=FALSE, mainClusterArgs=list(clusterFunction="pam", clusterArgs=list(k=8)))

#create dendrogram of clusters:
hcl <- makeDendrogram(cl)
plotDendrogram(hcl)
plotDendrogram(hcl, leafType="samples",plotType="colorblock")
```

---

mergeClusters *Merge clusters based on dendrogram*

Description

Takes an input of hierarchical clusterings of clusters and returns estimates of number of proportion of non-null and merges those below a certain cutoff.

Usage

```
## S4 method for signature 'matrixOrHDF5'
mergeClusters(
  x,
  cl,
  dendro = NULL,
  mergeMethod = c("none", "Storey", "PC", "adjP", "locfdr", "MB", "JC"),
  plotInfo = "none",
  nodePropTable = NULL,
  calculateAll = TRUE,
  showWarnings = FALSE,
  cutoff = 0.05,
  plot = TRUE,
  DEMethod,
  logFCcutoff = 0,
  weights = NULL,
  ...
)

## S4 method for signature 'ClusterExperiment'
mergeClusters(
  x,
  eraseOld = FALSE,
  mergeMethod = "none",
  plotInfo = "all",
  clusterLabel = "mergeClusters",
```

```
  leafType = c("samples", "clusters"),
  plotType = c("colorblock", "name", "ids"),
  plot = TRUE,
  whichAssay = 1,
  forceCalculate = FALSE,
  weights = if ("weights" %in% assayNames(x)) "weights" else NULL,
  DEMethod,
  ...
)

## S4 method for signature 'ClusterExperiment'
nodeMergeInfo(x)

## S4 method for signature 'ClusterExperiment'
mergeCutoff(x)

## S4 method for signature 'ClusterExperiment'
mergeMethod(x)

## S4 method for signature 'ClusterExperiment'
mergeClusterIndex(x)

## S4 method for signature 'ClusterExperiment'
eraseMergeInfo(x)

## S4 method for signature 'ClusterExperiment'
getMergeCorrespond(x, by = c("merge", "original"))
```

Arguments

x	data to perform the test on. It can be a matrix or a ClusterExperiment.
cl	A numeric vector with cluster assignments to compare to. "-1" indicates the sample was not assigned to a cluster.
dendro	dendrogram providing hierarchical clustering of clusters in cl. If x is a matrix, then the default is dendro=NULL and the function will calculate the dendrogram with the given (x, cl) pair using makeDendrogram. If x is a ClusterExperiment object, the dendrogram in the slot dendro_clusters will be used. In this case, this means that makeDendrogram needs to be called before mergeClusters.
mergeMethod	method for calculating proportion of non-null that will be used to merge clusters (if 'none', no merging will be done). See details for description of methods.
plotInfo	what type of information about the merging will be shown on the dendrogram. If 'all', then all the estimates of proportion non-null will be plotted at each node of the dendrogram; if 'mergeMethod', then only the value used in the mergeClusters command is plotted at each node. If 'none', then no proportions will be added to the dendrogram, though the dendrogram will be drawn. 'plotInfo' can also be one of the valid input to mergeMethod (even if that method is not the method chosen in mergeMethod argument). plotInfo can also show the information corresponding to "adjP" with a fold-change cutoff, by giving a value to this argument in the form of "adjP_2.0", for example.
nodePropTable	Only for matrix version. Matrix of results from previous run of mergeClusters as returned by matrix version of mergeClusters. Useful if just want to change the cutoff. Not generally intended for user but used internally by package.

`calculateAll` logical. Whether to calculate the estimates for all methods. This reduces computation costs for any future calls to `mergeClusters` since the results can be passed to future calls of `mergeClusters` (and for `ClusterExperiment` objects this is done automatically).

`showWarnings` logical. Whether to show warnings given by the methods. The 'locfdr' method in particular frequently spits out warnings (which may indicate that its estimates are not reliable). Setting `showWarnings=FALSE` will suppress all warnings from all methods (not just "locfdr"). By default this is set to `showWarnings=FALSE` by default to avoid large number of warnings being produced by "locfdr", but users may want to be more careful to check the warnings for themselves.

`cutoff` minimimum value required for NOT merging a cluster, i.e. two clusters with the proportion of DE below cutoff will be merged. Must be a value between 0, 1, where lower values will make it harder to merge clusters.

`plot` logical as to whether to plot the dendrogram with the merge results

`DEMethod` character vector describing how the differential expression analysis should be performed that will be used in the estimation of the percentage DE per node. See getBestFeatures for current options. See details.

`logFCcutoff` Relevant only if the `mergeMethod` selected is "adjP", in which case the calculation of the proportion of individual tests significant will also require that the estimated log-fold change of the features to be at least this large in absolute value. Value will be rounded to nearest tenth of an integer via `round(logFCcutoff,digits=1)`. For any other method, this parameter is ignored. Note that the logFC is based on `log2` (the results of `getBestFeatures`)

`weights` weights to use in by edgeR. If `x` is a matrix, then weights should be a matrix of weights, of the same dimensions as `x`. If `x` is a `ClusterExperiment` object `weights` can be a either a matrix, as previously described, or a character or numeric index to an assay in `x` that contains the weights. We recommend that weights be stored as an assay with name `"weights"` so that the weights will also be used with `mergeClusters`, and this is the default. Setting `weights=NULL` ensures that weights will NOT be used, and only the standard edgeR.

`...` for signature `matrix`, arguments passed to the `plot.phylo` function of ape that plots the dendrogram. For signature `ClusterExperiment` arguments passed to the method for signature `matrix` and then if do not match those arguments, will be passed onto `plot.phylo`.

`eraseOld` logical. Only relevant if input `x` is of class `ClusterExperiment`. If TRUE, will erase existing workflow results (clusterMany as well as mergeClusters and makeConsensus). If FALSE, existing workflow results will have `"_i"` added to the clusterTypes value, where `i` is one more than the largest such existing workflow clusterTypes.

`clusterLabel` a string used to describe the type of clustering. By default it is equal to "mergeClusters", to indicate that this clustering is the result of a call to mergeClusters (only if x is a ClusterExperiment object)

`leafType` if plotting, whether the leaves should be the clusters or the samples. Choosing 'samples' allows for visualization of how many samples are in the merged clusters (only if x is a ClusterExperiment object), which is the main difference between choosing "clusters" and "samples", particularly if `plotType="colorblock"`

`plotType` if plotting, then whether leaves of dendrogram should be labeled by rectangular blocks of color ("colorblock") or with the names of the leaves ("name") (only if x is a ClusterExperiment object).

whichAssay	numeric or character specifying which assay to use. See assay for details.
forceCalculate	This forces the function to erase previously saved merge results and recalculate the merging.
by	indicates whether output from getMergeCorrespond should be a vector/list with elements corresponding to merge cluster ids or elements corresponding to the original clustering ids. See return value for details.

Details

Estimation of proportion non-null "Storey" refers to the method of Storey (2002). "PC" refers to the method of Pounds and Cheng (2004). "JC" refers to the method of Ji and Cai (2007), and implementation of "JC" method is copied from code available on Jiashin Ji's website, December 16, 2015 (http://www.stat.cmu.edu/~jiashun/Research/software/NullandProp/). "locfdr" refers to the method of Efron (2004) and is implemented in the package locfdr. "MB" refers to the method of Meinshausen and Buhlmann (2005) and is implemented in the package howmany. "adjP" refers to the proportion of genes that are found significant based on a FDR adjusted p-values (method "BH") and a cutoff of 0.05.

Control of Plotting If mergeMethod is not equal to 'none' then the plotting will indicate where the clusters will be merged by making dotted lines of edges that are merged together (assuming plotInfo is not 'none'). plotInfo controls simultaneously what information will be plotted on the nodes as well as whether the dotted lines will be shown for the merged cluster. Notice that the choice of plotInfo (as long as it is not 'none') has no effect on how the dotted edges are drawn – they are always drawn based on the mergeMethod. If you choose plotInfo to not be equal to the mergeMethod, then you will have a confusing picture where the dotted edges will be based on the clustering created by mergeMethod while the information on the nodes is based on a different method. Note that you can override plotInfo by setting show.node.label=FALSE (passed to plot.phylo), so that no information is plotted on the nodes, but the dotted edges are still drawn. If you just want plot of the dendrogram, with no merging performed nor demonstrated on the plot, see plotDendrogram.

Saving and Reusing of results By default, the function saves the results in the ClusterExperiment object and will not recalculate them if not needed. Note that by default calculateAll=TRUE, which means that regardless of the value of mergeMethod, all the methods will be calculated so that those results will be stored and if you change the mergeMethod, no additional calculations are needed. Since the computationally intensive step is the running the DE method on the genes, this is a big savings (all of the methods then calculate the proportion from those results). However, note that if calculateAll=TRUE and ANY of the methods returned NA for any value, the calculation will be redone. Thus if, for example, the locfdr function does not run successfully and returns NA, the function will always recalculate each time, even if you don't specifically want the results of locfdr. In this case, it makes sense to turn calculateAll=FALSE.

If the dendrogram was made with option unassignedSamples="cluster" (i.e. unassigned were clustered in with other samples), then you cannot choose the option leafType='samples'. This is because the current code cannot reliably link up the internal nodes of the sample dendrogram to the internal nodes of the cluster dendrogram when the unassigned samples are intermixed.

When the input is a ClusterExperiment object, the function attempts to update the merge information in that object. This is done by checking that the existing dendrogram stored in the object (and run on the clustering stored in the slot dendro_index) is the same clustering that is stored in the slot merge_dendrocluster_index. For this reason, new calls to makeDendrogram will erase the merge information saved in the object.

If mergeClusters is run with mergeMethod="none", the function may still calculate the proportions per node if plotInfo is not equal to "none" or calculateAll=TRUE. If the input object was a ClusterExperiment object, the resulting information will be still saved, though no new clustering

was created; if there was not an existing merge method, the slot merge_dendrocluster_index will be updated.

Value

If 'x' is a matrix, it returns (invisibly) a list with elements

- clustering a vector of length equal to ncol(x) giving the integer-valued cluster ids for each sample. "-1" indicates the sample was not clustered.
- oldClToNew A table of the old cluster labels to the new cluster labels.
- nodeProp A table of the proportions that are DE on each node.This table is saved in the merge_nodeProp slot of a ClusterExperiment object and can be accessed along with the nodeMerge info with the nodeMergeInfo function.
- nodeMerge A table of indicating for each node whether merged or not and the cluster id in the new clustering that corresponds to the node. Note that a node can be merged and not correspond to a node in the new clustering, if its ancestor node is also merged. But there must be some node that corresponds to a new cluster id if merging has been done. This table is saved in the merge_nodeMerge slot of a ClusterExperiment object and can be accessed along with the nodeProp info with the nodeMergeInfo function.
- updatedClusterDendro The dendrogram on which the merging was based (based on the original clustering).
- cutoff The cutoff value for merging.

If 'x' is a ClusterExperiment, it returns a new ClusterExperiment object with an additional clustering based on the merging. This becomes the new primary clustering. Note that even if mergeMethod="none", the returned object will erase any old merge information, update the work flow numbering, and return the newly calculated merge information.

nodeMergeInfo returns information collected about the nodes during merging as a data.frame with the following entries:

- Node Name of the node
- ContrastThe contrast compared at each node, in terms of the cluster ids
- isMerged Logical as to whether samples from that node which were merged into one cluster during merging
- mergeClusterId If a node corresponds to a new, merged cluster, gives the cluster id it corresponds to. Otherwise NA
- ...The remaining columns give the estimated proportion of genes differentially expressed for each method. A column of NAs means that the method in question hasn't been calculated yet.

mergeCutoff returns the cutoff used for the current merging.

mergeMethod returns the method used for the current merge.

mergeClusterIndex returns the index of the clustering used for the current merge.

eraseMergeInfo returns object with all previously saved merge info removed.

getMergeCorrespond returns the correspondence between the merged cluster and its originating cluster. If by="original" returns a named vector, where the names of the vector are the cluster ids of the originating cluster and the values of the vector are the cluster ids of the merged cluster. If by="merge" the results returned are organized by the merged clusters. This will generally be a list, with the names of the list equal to the clusterIds of the merge clusters and the entries the clusterIds of the originating clusters. However, if there was no merging done (so that the clusters are identical) the output will be a vector like with by="original".

References

Ji and Cai (2007), "Estimating the Null and the Proportion of Nonnull Effects in Large-Scale Multiple Comparisons", JASA 102: 495-906.

Efron (2004) "Large-scale simultaneous hypothesis testing: the choice of a null hypothesis," JASA, 99: 96-104.

Meinshausen and Buhlmann (2005) "Lower bounds for the number of false null hypotheses for multiple testing of associations", Biometrika 92(4): 893-907.

Storey (2002) "A direct approach to false discovery rates", J. R. Statist. Soc. B 64 (3)": 479-498.

Pounds and Cheng (2004). "Improving false discovery rate estimation." Bioinformatics 20(11): 1737-1745.

See Also

makeDendrogram, plotDendrogram, getBestFeatures

Examples

```
data(simData)

#create a clustering, for 8 clusters (truth was 3)
cl<-clusterSingle(simData, subsample=FALSE,
sequential=FALSE, mainClusterArgs=list(clusterFunction="pam", clusterArgs=list(k=8)))

#give more interesting names to clusters:
newNames<- paste("Cluster",clusterLegend(cl)[[1]][,"name"],sep="")
clusterLegend(cl)[[1]][,"name"]<-newNames
#make dendrogram
cl <- makeDendrogram(cl)

#plot showing the before and after clustering
#(Note argument 'use.edge.length' can improve
#readability)
merged <- mergeClusters(cl, plotInfo="all",
mergeMethod="adjP", use.edge.length=FALSE, DEMethod="limma")

#Simpler plot with just dendrogram and single method
merged <- mergeClusters(cl, plotInfo="mergeMethod",
mergeMethod="adjP", use.edge.length=FALSE, DEMethod="limma",
leafType="clusters",plotType="name")

#compare merged to original
tableClusters(merged,whichClusters=c("mergeClusters","clusterSingle"))
```

---

numericalAsCharacter *Convert numeric values to character that sort correctly*

Description

Small function that takes as input integer values (or values that can be converted to integer values) and converts them into character values that are 'padded' with zeros at the beginning of the numbers so that they will sort correctly.

Usage

```
numericalAsCharacter(values, prefix = "")
```

Arguments

values	vector of values to be converted into sortable character values
prefix	optional character string that will be added as prefix to the result

Details

The function determines the largest value and adds zeros to the front of smaller integers so that the resulting characters are the same number of digits. This allows standard sorting of the values to correctly sort.

The maximum number of zeros that will be added is 3. Input integers beyond that point will not be correctly fixed for sorting.

Negative integers will not be corrected, but left as-is

Value

A character vector

See Also

str_pad

Examples

```
numericalAsCharacter(c(-1, 5,10,20,100))
```

---

plotBarplot,ClusterExperiment-method

Barplot of 1 or 2 clusterings

Description

Make a barplot of sample's assignments to clusters for single clustering, or cross comparison for two clusterings.

Usage

```
## S4 method for signature 'ClusterExperiment'
plotBarplot(object, whichClusters = "primary", labels = c("names", "ids"), ...)

## S4 method for signature 'vector'
plotBarplot(object, ...)

## S4 method for signature 'matrix'
plotBarplot(
  object,
  xNames = NULL,
  legNames = NULL,
```

```
  legend = ifelse(ncol(object) == 2, TRUE, FALSE),
  xlab = NULL,
  legend.title = NULL,
  unassignedColor = "white",
  missingColor = "grey",
  colPalette = NULL,
  ...
)
```

Arguments

object	A matrix of with each column corresponding to a clustering and each row a sample or a ClusterExperiment object.
whichClusters	argument that can be either numeric or character vector indicating the clusterings to be used. See details of getClusterIndex.
labels	if object is a ClusterExperiment object, then labels defines whether the clusters will be identified by their names values in clusterLegend (labels="names", the default) or by their clusterIds value in clusterLegend (labels="ids").
...	for plotBarplot arguments passed either to the method of plotBarplot for matrices or ultimately to barplot.
xNames	names for the clusters on x-axis (i.e. clustering given 1st). By default uses names of the 1st column of clusters matrix. See details.
legNames	names for the clusters dividing up the 1st clusters (will appear in legend). By default uses names of the 2nd cluster of clusters matrix. If only one clustering, xNames and legNames refer to the same clustering. See details.
legend	whether to plot the legend
xlab	label for x-axis. By default or if equal NULL the column name of the 1st cluster of clusters matrix
legend.title	label for legend. By default or if equal NULL the column name of the 2st cluster of clusters matrix
unassignedColor	If "-1" in clusters, will be given this color (meant for samples not assigned to cluster).
missingColor	If "-2" in clusters, will be given this color (meant for samples that were missing from the clustering, mainly when comparing clusterings run on different sets of samples)
colPalette	a vector of colors used for the different clusters. See details.

Details

The first column of the cluster matrix will be on the x-axis and the second column (if present) will separate the groups of the first column.

All arguments of the matrix version can be passed to the ClusterExperiment version. As noted above, however, some arguments have different interpretations.

If whichClusters = "workflow", then the most recent two clusters of the workflow will be chosen where recent is based on the following order (most recent first): final, mergeClusters, makeConsensus, clusterMany.

xNames, legNames and colPalette should all be named vectors, with the names referring to the clusters they should match to (for ClusterExperiment objects, it is determined by the argument

`labels` as to whether the names should match the cluster names or the clusterIds). `colPalette` and `legNames` must be same length of the number of clusters found in the second clustering, or if only a single clustering, the same length as the number of clusters in that clustering.

Value

A plot is produced, nothing is returned

Author(s)

Elizabeth Purdom

Examples

```
#clustering using pam: try using different dimensions of pca and different k
data(simData)

cl <- clusterMany(simData, nReducedDims=c(5, 10, 50), reduceMethod="PCA",
clusterFunction="pam", ks=2:4, findBestK=c(TRUE,FALSE),
removeSil=c(TRUE,FALSE), makeMissingDiss=TRUE)

plotBarplot(cl)
plotBarplot(cl,whichClusters=1:2)
```

`plotClusters` *Visualize cluster assignments across multiple clusterings*

Description

Align multiple clusterings of the same set of samples and provide a color-coded plot of their shared cluster assignments

Usage

```
## S4 method for signature 'ClusterExperiment'
plotClusters(
  object,
  whichClusters,
  existingColors = c("ignore", "all", "firstOnly"),
  resetNames = FALSE,
  resetColors = FALSE,
  resetOrderSamples = FALSE,
  colData = NULL,
  clusterLabels = NULL,
  ...
)

## S4 method for signature 'matrix'
plotClusters(
  object,
  orderSamples = NULL,
```

```
  colData = NULL,
  reuseColors = FALSE,
  matchToTop = FALSE,
  plot = TRUE,
  unassignedColor = "white",
  missingColor = "grey",
  minRequireColor = 0.3,
  startNewColors = FALSE,
  colPalette = massivePalette,
  input = c("clusters", "colors"),
  clusterLabels = colnames(object),
  add = FALSE,
  xCoord = NULL,
  ylim = NULL,
  tick = FALSE,
  ylab = "",
  xlab = "",
  axisLine = 0,
  box = FALSE,
  ...
)
```

Arguments

object	A matrix of with each column corresponding to a clustering and each row a sample or a ClusterExperiment object. If a matrix, the function will plot the clusterings in order of this matrix, and their order influences the plot greatly.
whichClusters	argument that can be either numeric or character vector indicating the clusterings to be used. See details of getClusterIndex.
existingColors	how to make use of the exiting colors in the ClusterExperiment object. 'ignore' will ignore them and assign new colors. 'firstOnly' will use the existing colors of only the 1st clustering, and then align the remaining clusters and give new colors for the remaining only. 'all' will use all of the existing colors.
resetNames	logical. Whether to reset the names of the clusters in clusterLegend to be the aligned integer-valued ids from plotClusters.
resetColors	logical. Whether to reset the colors in clusterLegend in the ClusterExperiment returned to be the colors from the plotClusters.
resetOrderSamples	logical. Whether to replace the existing orderSamples slot in the ClusterExperiment object with the new order found.
colData	If clusters is a matrix, colData gives a matrix of additional cluster/sample data on the samples to be plotted with the clusterings given in clusters. Values in colData will be added to the end (bottom) of the plot. NAs in the colData matrix will trigger an error. If clusters is a ClusterExperiment object, the input in colData refers to columns of the the colData slot of the ClusterExperiment object to be plotted with the clusters. In this case, colData can be TRUE (i.e. all columns will be plotted), or an index or a character vector that references a column or column name, respectively, of the colData slot of the ClusterExperiment object. If there are NAs in the colData columns, they will be encoded as 'unassigned' and receive the same color as 'unassigned' samples in the clustering.

clusterLabels	names to go with the columns (clusterings) in matrix colorMat. If colData argument is not NULL, the clusterLabels argument must include names for the sample data too. If the user gives only names for the clusterings, the code will try to anticipate that and use the column names of the sample data, but this is fragile. If set to FALSE, then no labels will be plotted.
...	for plotClusters arguments passed either to the method of plotClusters for matrices, or ultimately to plot (if add=FALSE).
orderSamples	A predefined order in which the samples will be plotted. Otherwise the order will be found internally by aligning the clusters (assuming input="clusters")
reuseColors	Logical. Whether each row should consist of the same set of colors. By default (FALSE) each cluster that the algorithm doesn't identify to the previous rows clusters gets a new color.
matchToTop	Logical as to whether all clusters should be aligned to the first row. By default (FALSE) each cluster is aligned to the ordered clusters of the row above it.
plot	Logical as to whether a plot should be produced.
unassignedColor	If "-1" in clusters, will be given this color (meant for samples not assigned to cluster).
missingColor	If "-2" in clusters, will be given this color (meant for samples that were missing from the clustering, mainly when comparing clusterings run on different sets of samples)
minRequireColor	In aligning colors between rows of clusters, require this percent overlap.
startNewColors	logical, indicating whether in aligning colors between rows of clusters, should the colors restart at beginning of colPalette as long as colors are not in immediately proceeding row (the colors at the end of massivePalette are all of colors() and many will be indistinguishable, so this option can be useful if you have a large cluster matrix).
colPalette	a vector of colors used for the different clusters. Must be as long as the maximum number of clusters found in any single clustering/column given in clusters or will otherwise return an error.
input	indicate whether the input matrix is matrix of integer assigned clusters, or contains the colors. If input="colors", then the object clusters is a matrix of colors and there is no alignment (this option allows the user to manually adjust the colors and replot, for example).
add	whether to add to existing plot.
xCoord	values on x-axis at which to plot the rows (samples).
ylim	vector of limits of y-axis.
tick	logical, whether to draw ticks on x-axis for each sample.
ylab	character string for the label of y-axis.
xlab	character string for the label of x-axis.
axisLine	the number of lines in the axis labels on y-axis should be (passed to line = ... in the axis call).
box	logical, whether to draw box around the plot.

Details

All arguments of the matrix version can be passed to the ClusterExperiment version. As noted above, however, some arguments have different interpretations.

If whichClusters = "workflow", then the workflow clusterings will be plotted in the following order: final, mergeClusters, makeConsensus, clusterMany.

Value

If clusters is a ClusterExperiment Object, then plotClusters returns an updated ClusterExperiment object, where the clusterLegend and/or orderSamples slots have been updated (depending on the arguments).

If clusters is a matrix, plotClusters returns (invisibly) the orders and other things that go into making the matrix. Specifically, a list with the following elements.

- orderSamples a vector of length equal to nrows(clusters) giving the order of the samples (rows) to use to get the original clusters matrix into the order made by plotClusters.
- colors matrix of color assignments for each element of original clusters matrix. Matrix is in the same order as original clusters matrix. The matrix colors[orderSamples,] is the matrix that can be given back to plotClusters to recreate the plot (see examples).
- alignedClusterIds a matrix of integer valued cluster assignments that match the colors. This is useful if you want to have cluster identification numbers that are better aligned than that given in the original clusters. Again, the rows/samples are in same order as original input matrix.
- clusterLegend list of length equal to the number of columns of input matrix. The elements of the list are matrices, each with three columns named "Original","Aligned", and "Color" giving, respectively, the correspondence between the original cluster ids in clusters, the aligned cluster ids in aligned, and the color.
- origClustersThe original matrix of clusters given to plotClusters

Author(s)

Elizabeth Purdom and Marla Johnson (based on the tracking plot in ConsensusClusterPlus by Matt Wilkerson and Peter Waltman).

References

Wilkerson, D. M, Hayes and Neil D (2010). "ConsensusClusterPlus: a class discovery tool with confidence assessments and item tracking." Bioinformatics, 26(12), pp. 1572-1573.

See Also

The ConsensusClusterPlus package.

Examples

```
#clustering using pam: try using different dimensions of pca and different k
data(simData)

cl <- clusterMany(simData, nReducedDims=c(5, 10, 50), reduceMethod="PCA",
clusterFunction="pam", ks=2:4, findBestK=c(TRUE,FALSE),
removeSil=c(TRUE,FALSE), makeMissingDiss=TRUE)
```

```
clusterLabels(cl)

#make names shorter for better plotting
x <- clusterLabels(cl)
x <- gsub("TRUE", "T", x)
x <- gsub("FALSE", "F", x)
x <- gsub("k=NA,", "", x)
x <- gsub("Features", "", x)
clusterLabels(cl) <- x

par(mar=c(2,10,1,1))
#this will make the choices of plotClusters
cl <- plotClusters(cl, axisLine=-1, resetOrderSamples=TRUE, resetColors=TRUE)

#see the new cluster colors
clusterLegend(cl)[1:2]

#We can also change the order of the clusterings. Notice how this
#dramatically changes the plot!
clOrder <- c(3:6, 1:2, 7:ncol(clusterMatrix(cl)))
cl <- plotClusters(cl, whichClusters=clOrder, resetColors=TRUE,
resetOrder=TRUE, axisLine=-2)

#We can manually switch the red ("#E31A1C") and green ("#33A02C") in the
#first cluster:

#see what the default colors are and their names
showPalette(wh=1:5)

#change "#E31A1C" to "#33A02C"
newColorMat <- clusterLegend(cl)[[clOrder[1]]]
newColorMat[2:3, "color"] <- c("#33A02C", "#E31A1C")
clusterLegend(cl)[[clOrder[1]]]<-newColorMat

#replot by setting 'input="colors"'
par(mfrow=c(1,2))
plotClusters(cl, whichClusters=clOrder, orderSamples=orderSamples(cl),
existingColors="all")
plotClusters(cl, whichClusters=clOrder, resetColors=TRUE, resetOrder=TRUE,
axisLine=-2)
par(mfrow=c(1,1))

#set some of clusterings arbitrarily to "-1", meaning not clustered (white),
#and "-2" (another possible designation getting gray, usually for samples not
#included in original clustering)
clMatNew <- apply(clusterMatrix(cl), 2, function(x) {
wh <- sample(1:nSamples(cl), size=10)
x[wh]<- -1
wh <- sample(1:nSamples(cl), size=10)
x[wh]<- -2
return(x)
})

#make a new object
cl2 <- ClusterExperiment(assay(cl), clMatNew,
transformation=transformation(cl))
plotClusters(cl2)
```

plotClustersTable *Plot heatmap of cross-tabs of 2 clusterings*

Description

Plot heatmap of cross-tabulations of two clusterings

Usage

```
## S4 method for signature 'ClusterExperiment'
plotClustersTable(
  object,
  whichClusters,
  ignoreUnassigned = FALSE,
  margin = NA,
  ...
)

## S4 method for signature 'table'
plotClustersTable(
  object,
  plotType = c("heatmap", "bubble"),
  main = "",
  xlab = NULL,
  ylab = NULL,
  legend = TRUE,
  cluster = FALSE,
  clusterLegend = NULL,
  sizeTable = NULL,
  ...
)

## S4 method for signature 'ClusterExperiment'
tableClusters(
  object,
  whichClusters = "primary",
  useNames = TRUE,
  tableMethod = c("intersect", "union"),
  ...
)

## S4 method for signature 'table,table'
bubblePlot(
  propTable,
  sizeTable,
  gridColor = rgb(0, 0, 0, 0.05),
  maxCex = 8,
  cexFactor,
  ylab,
  xlab,
  propLabel = "Value of %",
```

```
  legend = TRUE,
  las = 2,
  colorScale = RColorBrewer::brewer.pal(11, "Spectral")[-6]
)
```

Arguments

object	ClusterExperiment object (or matrix with table result)
whichClusters	argument that can be either numeric or character vector indicating the clusterings to be used. See details of getClusterIndex.
ignoreUnassigned	logical as to whether to ignore unassigned clusters in the plotting. This means they will also be ignored in the calculations of the proportions (if margin not NA).
margin	if NA, the actual counts from tableClusters will be plotted. Otherwise, prop.table will be called and the argument margin will be passed to prop.table to determine whether proportions should be calculated. If '1', then the proportions in the rows sum to 1, if '2' the proportions in the columns sum to 1. If 'NULL' then the proportion across the entire matrix will sum to 1. An additional option has been added so that if you set margin=0, the entry displayed in each cell will be the proportion equal to the size of the intersection over the size of the union of the clusters (a Jaccard similarity between the clusters), in which case each entry is a value between 0 and 1 but no combination of the entries sum to 1.
...	arguments passed on to plotHeatmap or bubblePlot depending on choice of plotType. Note that these functions take different arguments so that switching from one to the other may not take all arguments. In particular bubblePlot calls plot while plotHeatmap calls NMF{aheatmap}.
plotType	type of plot. If "heatmap", then a heatmap will be created of the values of the contingency table of the two clusters (calculated as determined by the argument "margin") using plotHeatmap. If "bubble", then a plot will be created using bubblePlot, which will create circles for each cell of the contingencey table whose size corresponds to the number of samples shared and the color based on the value of the proportion (as chosen by the argument margin).
main	title of plot, passed to plotHeatmap or to the argument propLabel in bubblePlot
xlab	label for labeling clustering on the x-axis. If NULL, will determine names. If set to NA no label for clustering on the x-axis will be plotted (to turn off legend of the clusterings in heatmap, set legend=FALSE).
ylab	label for labeling clustering on the y-axis. If NULL, will determine names. If set to NA no label for clustering on the y-axis will be plotted (to turn off legend of the clusterings in heatmap, set legend=FALSE).
legend	whether to draw legend along top (bubble plot) or the color legend (heatmap)
cluster	logical, whether to cluster the rows and columns of the table. Passed to arguments clusterFeatures AND clusterSamples of plotHeatmap.
clusterLegend	list in clusterLegend format that gives colors for the clusters tabulated.
sizeTable	table of sizes (only for use in bubblePlot or plotType="bubble"). See details.
useNames	for tableClusters, whether the output should be tabled with names (useNames=TRUE) or ids (useNames=FALSE)

tableMethod	the type of table calculation to perform. "intersect" refers to the standard contingency table (table), where each entry of the resulting table is the number of objects in both clusters. "union" instead gives for each entry the number of objects that are in the union of both clusters.
propTable	table of proportions (bubblePlot))
gridColor	color for grid lines (bubblePlot))
maxCex	largest value of cex for any point (others will scale proportionally smaller) (bubblePlot)).
cexFactor	factor to multiple by to get values of circles. If missing, finds value automatically, namely by using the maxCex value default. Overrides value of maxCex. (bubblePlot))
propLabel	the label to go with the legend of the color of the bubbles/circles
las	the value for the las value in the call to axis in labeling the clusters in the bubble plot. Determines whether parallel or perpindicular labels to the axis (see par).
colorScale	the color scale for the values of the proportion table

Details

For plotClustersTable applied to the class table, sizeTable is passed to bubblePlot to indicate the size of the circle. If sizeTable=NULL, then it is assumed that the object argument is the table of counts and both the propTable and sizeTable are set to the same value (hence turning off the coloring of the circle/bubbles). This is equivalent effect to the margin=NA option of plotClustersTable applied to the ClusterExperiment class.

Note that the cluster labels in plotClustersTable and tableClusters are converted to "proper" R names via make.names. This is because tableClusters calls the R function table, which makes this conversion

For plotClustersTable, whichClusters should define 2 clusters, while for tableClusters it can indicate arbitrary number.

bubblePlot is mainly used internally by plotClustersTable but is made public for users who want more control and to allow documentation of the arguments. bubblePlot plots a circle for each intersection of two clusters, where the color of the circle is based on the value in propTable and the size of the circle is based on the value in sizeTable. If propTable is equal to sizeTable, then the propTable is ignored and the coloring of the circles is not performed, only the adjusting of the size of the circles based on the total size. The size is determined by setting the cex value of the point as $sqrt(sizeTable[i,j])/sqrt(max(sizeTable))*cexFactor$.

Value

tableClusters returns an object of class table (see table).

plotClustersTables returns invisibly the plotted proportion table. In particular, this is the result of applying prop.table to the results of tableClusters (after removing unclustered samples if ignoreUnassigned=TRUE).

Author(s)

Kelly Street, Elizabeth Purdom

See Also

plotHeatmap

table

prop.table

Examples

```
#clustering using pam: try using different dimensions of pca and different k
data(simData)

cl <- clusterMany(simData, nReducedDims=c(5, 10, 50), reducedDim="PCA",
clusterFunction="pam", ks=2:4, findBestK=c(TRUE,FALSE),
removeSil=c(TRUE,FALSE), makeMissingDiss=TRUE)
#give arbitrary names to clusters for demonstration
cl<-renameClusters(cl,value=letters[1:nClusters(cl)[1]],whichCluster=1)
tableClusters(cl,whichClusters=1:2)
#show options of margin in heatmap format:
par(mfrow=c(2,3))
plotClustersTable(cl,whichClusters=1:2, margin=NA, legend=FALSE,
  ignoreUnassigned=TRUE)
plotClustersTable(cl,whichClusters=1:2, margin=0, legend=FALSE,
  ignoreUnassigned=TRUE)
plotClustersTable(cl,whichClusters=1:2, margin=1, legend=FALSE,
  ignoreUnassigned=TRUE)
plotClustersTable(cl,whichClusters=1:2, margin=2, legend=FALSE,
  ignoreUnassigned=TRUE)
plotClustersTable(cl,whichClusters=1:2, margin=NULL, legend=FALSE,
  ignoreUnassigned=TRUE)

#show options of margin in bubble format:
par(mfrow=c(2,3))
plotClustersTable(cl,whichClusters=1:2, margin=NA,
   ignoreUnassigned=TRUE, plotType="bubble")
plotClustersTable(cl,whichClusters=1:2, margin=0,
   ignoreUnassigned=TRUE, plotType="bubble")
plotClustersTable(cl,whichClusters=1:2, margin=1,
   ignoreUnassigned=TRUE, plotType="bubble")
plotClustersTable(cl,whichClusters=1:2, margin=2,
   ignoreUnassigned=TRUE, plotType="bubble")
plotClustersTable(cl,whichClusters=1:2, margin=NULL,
   ignoreUnassigned=TRUE, plotType="bubble")
```

---

plotClustersWorkflow,ClusterExperiment-method

A plot of clusterings specific for clusterMany and workflow visualization

---

Description

A realization of plotClusters call specific to separating out the results of clusterMany and other clustering results.

Usage

```
## S4 method for signature 'ClusterExperiment'
plotClustersWorkflow(
  object,
  whichClusters = c("mergeClusters", "makeConsensus"),
  whichClusterMany = NULL,
  nBlankLines = ceiling(nClusterings(object) * 0.05),
  existingColors = c("ignore", "all", "highlightOnly"),
  nSizeResult = ceiling(nClusterings(object) * 0.02),
  clusterLabels = TRUE,
  clusterManyLabels = TRUE,
  sortBy = c("highlighted", "clusterMany"),
  highlightOnTop = TRUE,
  ...
)
```

Arguments

object	A ClusterExperiment object on which clusterMany has been run
whichClusters	which clusterings to "highlight", i.e draw separately from the bulk of the plot, see argument whichClusters of getClusterIndex for description of format allowed.
whichClusterMany	indicate which clusterings to plot in the bulk of the plot, see argument whichClusters of getClusterIndex for description of format allowed.
nBlankLines	the number of blank (i.e. white) rows to add between the clusterMany clusterings and the highlighted clusterings.
existingColors	one of "ignore","all","highlightOnly". Whether the plot should use the stored colors in the ClusterExperiment object given. "highlightOnly" means only the highlighted clusters will use the stored colors, not the clusterMany clusterings.
nSizeResult	the number of rows each highlighted clustering should take up. Increasing the number increases the thickness of the rectangles representing the highlighted clusterings.
clusterLabels	either logical, indicating whether to plot the labels for the clusterings identified to be highlighted in the whichClusters argument, or a character vector of labels to use.
clusterManyLabels	either logical, indicating whether to plot the labels for the clusterings from clusterMany identified in the whichClusterMany, or a character vector of labels to use.
sortBy	how to align the clusters. If "highlighted" then the highlighted clusters indicated in the argument whichClusters are first in the alignment done by plotClusters. If "clusterMany", then the clusterMany results are first in the alignment. (Note this does not determine where they will be plotted, but how they are ordered in the aligning step done by plotClusters)
highlightOnTop	logical. Whether the highlighted clusters should be plotted on the top of clusterMany results or underneath.
...	arguments passed to the matrix version of plotClusters

Details

This plot is solely intended to make it easier to use the plotClusters visualization when there are a large number of clusterings from a call to clusterMany. This plot separates out the clusterMany results from a designated clustering of interest, as indicated by the whichClusters argument (by default clusterings from a call to makeConsensus or mergeClusters). In addition the highlighted clusters are made bigger so that they can be easily seen.

Value

A plot is produced, nothing is returned.

See Also

plotClusters, clusterMany

Examples

```
#clustering using pam: try using different dimensions of pca and different k
data(simData)

cl <- clusterMany(simData, nReducedDims=c(5, 10, 50), reduceMethod="PCA",
clusterFunction="pam", ks=2:4, findBestK=c(TRUE,FALSE),
removeSil=c(TRUE,FALSE), makeMissingDiss=TRUE)
cl <- makeConsensus(cl, proportion=0.7)
plotClustersWorkflow(cl)
```

---

plotContrastHeatmap,ClusterExperiment-method

Plot heatmaps showing significant genes per contrast

Description

Plots a heatmap of the data, with the genes grouped based on the contrast for which they were significant.

Usage

```
## S4 method for signature 'ClusterExperiment'
plotContrastHeatmap(
  object,
  signifTable,
  whichCluster = NULL,
  contrastColors = NULL,
  ...
)
```

Arguments

object	ClusterExperiment object on which biomarkers were found
signifTable	A data.frame in format of the result of getBestFeatures. It must minimally contain columns 'Contrast' and 'IndexInOriginal' giving the grouping and original index of the features in the assay(object)

whichCluster	if not NULL, indicates cluster used in making the significance table. Used to match to colors in clusterLegend(object) (relevant for one-vs-all contrast so that color aligns). See description of argument in getClusterIndex for futher details.
contrastColors	vector of colors to be given to contrasts. Should match the name of the contrasts in the 'Contrast' column of signifTable or 'ContrastName', if given.. If missing, default colors given by match to the cluster names of whichCluster (see above), or otherwise given a default assignment.
...	Arguments passed to plotHeatmap

Details

If the column 'ContrastName' is given in signifTable, these names will be used to describe the contrast in the legend.

Within each contrast, the genes are sorted by log fold-change if the column "logFC" is in the signifTable data.frame

Note that if whichCluster is NOT given (the default) then there is no automatic match of colors with contrasts based on the information in object.

Value

A heatmap is created. The output of plotHeatmap is returned.

See Also

plotHeatmap, makeBlankData, getBestFeatures

Examples

```
data(simData)

cl <- clusterSingle(simData, subsample=FALSE,
sequential=FALSE,
mainClusterArgs=list(clusterFunction="pam", clusterArgs=list(k=8)))

#Do all pairwise, only return significant, try different adjustments:
pairsPerC <- getBestFeatures(cl, contrastType="Pairs", number=5,
p.value=0.05, DEMethod="limma")
plotContrastHeatmap(cl,pairsPerC)
```

---

plotDendrogram,ClusterExperiment-method

Plot dendrogram of ClusterExperiment object

Description

Plots the dendrogram saved in a ClusterExperiment object

Usage

```
## S4 method for signature 'ClusterExperiment'
plotDendrogram(
  x,
  whichClusters = "dendro",
  leafType = c("samples", "clusters"),
  plotType = c("colorblock", "name", "ids"),
  mergeInfo = "none",
  main,
  sub,
  clusterLabelAngle = 45,
  removeOutbranch = TRUE,
  legend = c("side", "below", "none"),
  nodeColors = NULL,
  colData = NULL,
  clusterLegend = NULL,
  ...
)
```

Arguments

x	a ClusterExperiment object.
whichClusters	argument that can be either numeric or character vector indicating the clusterings to be used. See details of getClusterIndex.
leafType	if "samples" the dendrogram has one leaf per sample, otherwise it has one per cluster.
plotType	one of 'name', 'colorblock' or 'id'. If 'Name' then dendrogram will be plotted, and name of cluster or sample (depending on type of value for leafType) will be plotted next to the leaf of the dendrogram. If 'colorblock', rectangular blocks, corresponding to the color of the cluster will be plotted, along with cluster name legend. If 'id' the internal clusterIds value will be plotted (only appropriate if leafType="clusters").
mergeInfo	What kind of information about merge to plot on dendrogram. If not equal to "none", will replicate the kind of plot that mergeClusters creates, and the input to mergeInfo corresponds to that of plotInfo in mergeClusters.
main	passed to the plot.phylo function to set main title.
sub	passed to the plot.phylo function to set subtitle.
clusterLabelAngle	angle at which label of cluster will be drawn. Only applicable if plotType="colorblock".
removeOutbranch	logical, only applicable if there are missing samples (i.e. equal to -1 or -2), leafType="samples" and the dendrogram for the samples was made by putting missing samples in an outbranch. In which case, if this parameter is TRUE, the outbranch will not be plotted, and if FALSE it will be plotted.
legend	character, only applicable if plotType="colorblock". Passed to phydataplot in ape package that is used to draw the color values of the clusters/samples next to the dendrogram. Options are 'none', 'below', or 'side'. (Note 'none' is only available for 'ape' package >= 4.1-0.6).

nodeColors	named vector of colors to be plotted on a node in the dendrogram (calls nodelabels). Names should match the *internal* name of the node (the "NodeId" value, see clusterDendrogram).
colData	index (by integer or name) the sample data stored as a DataFrame in colData slot of the object. Only discrete valued ("character" or "factor" variables) will be plotted; indexing of continous variables will be ignored. Whether that data is continuous or not will be determined by the properties of colData (no user input is needed). This argument is only relevant if plotType=="colorblock" and leafType=="samples"
clusterLegend	Assignment of colors to the clusters or sample data (as designated by colData argument) plotted with the dendrogram . If NULL or a particular variable/cluster are not assigned a color, colors will be assigned internally for sample data and pull from the clusterLegend slot of the x for the clusters.
...	arguments passed to the plot.phylo function of ape that plots the dendrogram.

Details

If leafType="clusters", the plotting function will work best if the clusters in the dendrogram correspond to the primary cluster. This is because the function colors the cluster labels based on the colors of the clusterIds of the primaryCluster

Value

A dendrogram is plotted. Returns (invisibly) a list with elements

- plottedObject the phylo object that is plotted.
- originalObject the phylo object before adjusting the node/tip labels.

See Also

mergeClusters,plot.phylo, nodelabels,tiplabels

Examples

```
data(simData)

#create a clustering, for 8 clusters (truth was 3)
cl <-clusterSingle(simData, subsample=FALSE,
sequential=FALSE,
mainClusterArgs=list(clusterFunction="pam", clusterArgs=list(k=8)))

#create dendrogram of clusters and then
# merge clusters based ondendrogram:
cl <- makeDendrogram(cl)
cl <- mergeClusters(cl,mergeMethod="adjP",DEMethod="limma",
   cutoff=0.1,plot=FALSE)
plotDendrogram(cl)
plotDendrogram(cl,leafType="samples",whichClusters="all",plotType="colorblock")
```

`plotFeatureBoxplot` *Plot boxplot of feature values by cluster*

Description

Plot a boxplot of the (transformed) values for a particular gene, separated by cluster

Usage

```
## S4 method for signature 'ClusterExperiment,character'
plotFeatureBoxplot(object, feature, whichCluster = "primary", ...)

## S4 method for signature 'ClusterExperiment,numeric'
plotFeatureBoxplot(
  object,
  feature,
  whichCluster = "primary",
  plotUnassigned = FALSE,
  unassignedColor = NULL,
  missingColor = NULL,
  main = NULL,
  whichAssay = 1,
  ...
)
```

Arguments

`object`	a ClusterExperiment object
`feature`	identification of feature to plot, either row name or index
`whichCluster`	argument that can be a single numeric or character value indicating the *single* clustering to be used. Giving values that result in more than one clustering will result in an error. See details of `getClusterIndex`.
`...`	arguments passed to `boxplot`
`plotUnassigned`	whether to plot the unassigned samples as a cluster (either -1 or -2)
`unassignedColor`	If not NULL, should be character value giving the color for unassigned (-2) samples (overrides `clusterLegend`) default.
`missingColor`	If not NULL, should be character value giving the color for missing (-2) samples (overrides `clusterLegend`) default.
`main`	title of plot. If NULL, given default title.
`whichAssay`	numeric or character specifying which assay to use. See `assay` for details.

Value

A plot is created. The output of boxplot is returned (see `boxplot`), with additional elements `colors` and `clusterIds` that gives the colors and internal ids that match each boxplot (pulled from `clusterLegend` but in the order of plot)

See Also

boxplot

Examples

```
data(simData)
#Create a ClusterExperiment object
cl <- clusterMany(simData, nReducedDims=c(5, 10, 50), reducedDim="PCA",
   clusterFunction="pam", ks=2:4, findBestK=c(TRUE,FALSE),
   removeSil=c(TRUE,FALSE), makeMissingDiss=TRUE)
#give names to the clusters
cl<-renameClusters(cl, whichCluster=1,
   value=letters[1:nClusters(cl)[1]])
plotFeatureBoxplot(cl,feature=1)
```

---

plotFeatureScatter *Plot scatter plot of feature values colored by cluster*

Description

Plot a scatter plot of the (transformed) values for a set of gene expression values, colored by cluster

Usage

```
## S4 method for signature 'ClusterExperiment,character'
plotFeatureScatter(object, features, ...)

## S4 method for signature 'ClusterExperiment,numeric'
plotFeatureScatter(
  object,
  features,
  whichCluster = "primary",
  plotUnassigned = TRUE,
  unassignedColor = "grey",
  missingColor = "white",
  whichAssay = 1,
  legendLocation = NA,
  jitterFactor = NA,
  ...
)
```

Arguments

object	a ClusterExperiment object
features	the indices of the features (either numeric or character matching rownames of object) to be plotted.
...	arguments passed to boxplot
whichCluster	argument that can be a single numeric or character value indicating the *single* clustering to be used. Giving values that result in more than one clustering will result in an error. See details of getClusterIndex.

plotUnassigned	whether to plot the unassigned samples as a cluster (either -1 or -2)
unassignedColor	If not NULL, should be character value giving the color for unassigned (-2) samples (overrides clusterLegend) default.
missingColor	If not NULL, should be character value giving the color for missing (-2) samples (overrides clusterLegend) default.
whichAssay	numeric or character specifying which assay to use. See assay for details.
legendLocation	character value passed to location argument of plotClusterLegend indicating where to put the legend. If NA, legend will not be plotted.
jitterFactor	numeric. If NA, no jittering is done. Otherwise, passed to factor of function jitter (useful for low counts)

Value

returns invisibly the results of pairs or plot command.

Examples

```
data(simData)
#Create a ClusterExperiment object
cl <- clusterMany(simData, nReducedDims=c(5, 10, 50), reducedDim="PCA",
   clusterFunction="pam", ks=2:4, findBestK=c(TRUE,FALSE),
   removeSil=c(TRUE,FALSE), makeMissingDiss=TRUE)
#give names to the clusters
cl<-renameClusters(cl, whichCluster=1,
   value=letters[1:nClusters(cl)[1]])
plotFeatureScatter(cl,feature=1:2,pch=19,legendLocation="topright")
```

---

plotHeatmap *Heatmap for showing clustering results and more*

Description

Make heatmap with color scale from one matrix and hiearchical clustering of samples/features from another. Also built in functionality for showing the clusterings with the heatmap. Builds on aheatmap function of NMF package.

Usage

```
## S4 method for signature 'SingleCellExperiment'
plotHeatmap(data, isCount = FALSE, transFun = NULL, ...)

## S4 method for signature 'SummarizedExperiment'
plotHeatmap(data, isCount = FALSE, transFun = NULL, ...)

## S4 method for signature 'table'
plotHeatmap(data, ...)

## S4 method for signature 'ClusterExperiment'
plotHeatmap(
  data,
```

```
  clusterSamplesData = c("dendrogramValue", "hclust", "orderSamplesValue",
    "primaryCluster"),
  clusterFeaturesData = "var",
  nFeatures = NA,
  visualizeData = c("transformed", "centeredAndScaled", "original"),
  whichClusters = c("primary", "workflow", "all", "none"),
  colData = NULL,
  clusterFeatures = TRUE,
  nBlankLines = 2,
  colorScale,
  whichAssay = 1,
  ...
)

## S4 method for signature 'data.frame'
plotHeatmap(data, ...)

## S4 method for signature 'ExpressionSet'
plotHeatmap(data, ...)

## S4 method for signature 'matrixOrHDF5'
plotHeatmap(
  data,
  colData = NULL,
  clusterSamplesData = NULL,
  clusterFeaturesData = NULL,
  whColDataCont = NULL,
  clusterSamples = TRUE,
  showSampleNames = FALSE,
  clusterFeatures = TRUE,
  showFeatureNames = FALSE,
  colorScale = seqPal5,
  clusterLegend = NULL,
  alignColData = FALSE,
  unassignedColor = "white",
  missingColor = "grey",
  breaks = NA,
  symmetricBreaks = FALSE,
  capBreaksLegend = FALSE,
  isSymmetric = FALSE,
  overRideClusterLimit = FALSE,
  plot = TRUE,
  labelTracks = TRUE,
  ...
)

## S4 method for signature 'ClusterExperiment'
plotCoClustering(data, invert, saveDistance = FALSE, ...)
```

Arguments

data	data to use to determine the heatmap. Can be a matrix, ClusterExperiment, SingleCellExperiment or SummarizedExperiment object. The interpretation

of parameters depends on the type of the input to data.

isCount — if transFun=NULL, then isCount=TRUE will determine the transformation as defined by function(x){log2(x+1)}, and isCount=FALSE will give a transformation function function(x){x}. Ignored if transFun=NULL. If object is of class ClusterExperiment, the stored transformation will be used and giving this parameter will result in an error.

transFun — a transformation function to be applied to the data. If the transformation applied to the data creates an error or NA values, then the function will throw an error. If object is of class ClusterExperiment, the stored transformation will be used and giving this parameter will result in an error.

... — for signature matrix, arguments passed to aheatmap. For the other signatures, passed to the method for signature matrix. Not all arguments can be passed to aheatmap effectively, see details.

clusterSamplesData — If data is a matrix, clusterSamplesData is either a matrix that will be used by hclust to define the hiearchical clustering of samples (e.g. normalized data) or a pre-existing dendrogram (of class dendrogram) that clusters the samples. If data is a ClusterExperiment object, clusterSamplesData should be either character or integers or logical which indicates how (and whether) the samples should be clustered (or gives indices of the order for the samples). See details.

clusterFeaturesData — If data is a matrix, either a matrix that will be used in hclust to define the hiearchical clustering of features (e.g. normalized data) or a pre-existing dendrogram that clusters the features. If data is a ClusterExperiment object, the input should be either character or integers indicating which features should be used (see details).

nFeatures — integer indicating how many features should be used (if clusterFeaturesData is 'var' or 'PCA').

visualizeData — either a character string, indicating what form of the data should be used for visualizing the data (i.e. for making the color-scale), or a data.frame/matrix with same number of samples as assay(data). If a new data.frame/matrix, any character arguments to clusterFeaturesData will be ignored.

whichClusters — argument that can be either numeric or character vector indicating the clusterings to be used. See details of getClusterIndex.

colData — If input to data is either a ClusterExperiment,or SummarizedExperiment object or SingleCellExperiment, then colData must index the colData stored as a DataFrame in colData slot of the object. Whether that data is continuous or not will be determined by the properties of colData (no user input is needed). If input to data is matrix, colData is a matrix of additional data on the samples to show above heatmap. In this case, unless indicated by whColDataCont, colData will be converted into factors, even if numeric. "-1" indicates the sample was not assigned to a cluster and gets color 'unassignedColor' and "-2" gets the color 'missingColor'.

clusterFeatures — Logical as to whether to do hiearchical clustering of features (if FALSE, any input to clusterFeaturesData is ignored).

nBlankLines — Only applicable if input is ClusterExperiment object. Indicates the number of lines to put between groups of features if clusterFeaturesData gives groups of genes (see details and makeBlankData).

`colorScale` palette of colors for the color scale of the heatmap.

`whichAssay` numeric or character specifying which assay to use. See `assay` for details.

`whColDataCont` Which of the `colData` columns are continuous and should not be converted to counts. `NULL` indicates no additional `colData`. Only used if `data` input is matrix.

`clusterSamples` Logical as to whether to do hierarchical clustering of cells (if FALSE, any input to clusterSamplesData is ignored).

`showSampleNames`
Logical as to whether show sample names.

`showFeatureNames`
Logical as to whether show feature names.

`clusterLegend` Assignment of colors to the clusters. If `NULL`, `colData` columns will be assigned colors internally. See details for more.

`alignColData` Logical as to whether should align the colors of the `colData` (only if `clusterLegend` not given and `colData` is not `NULL`).

`unassignedColor`
color assigned to cluster values of '-1' ("unassigned").

`missingColor` color assigned to cluster values of '-2' ("missing").

`breaks` Either a vector of breaks (should be equal to length 52), or a number between 0 and 1, indicating that the breaks should be equally spaced (based on the range in the data) upto the 'breaks' quantile, see `setBreaks`

`symmetricBreaks`
logical as to whether the breaks created for the color scale should be symmetrical around 0

`capBreaksLegend`
logical as to whether the legend for the breaks should be capped. Only relevant if `breaks` is a value < 1, in which case if `capBreaksLegend=TRUE`, only the values between the quantiles requested will show in the color scale legend.

`isSymmetric` logical. if TRUE indicates that the input matrix is symmetric. Useful when plotting a co-clustering matrix or other sample by sample matrices (e.g., correlation).

`overRideClusterLimit`
logical. Whether to override the internal limit that only allows 10 clusterings/annotations. If overridden, may result in incomprehensible errors from `aheatmap`. Only override this if you have a very large plotting device and want to see if `aheatmap` can render it.

`plot` logical indicating whether to plot the heatmap. Mainly useful for package mantaince to avoid calls to aheatmap on unit tests that take a long time.

`labelTracks` logical, whether to put labels next to the color tracks corresponding to the colData.

`invert` logical determining whether the coClustering matrix should be inverted to be 1-coClustering for plotting. By default, if the diagonal elements are all zero, invert=TRUE, and otherwise invert=FALSE. If coClustering matrix is not a 0-1 matrix (e.g. if equal to a distance matrix output from `clusterSingle`, then the user should manually set this parameter to FALSE.)

`saveDistance` logical. When the `coClustering` slot contains indices of the clusterings or a NxB set of clusterings, the hamming distance will be calculated before running the plot. This argument determines whether the `ClusterExperiment` object with that distance in `coClustering` slot should be returned (so as to avoid recalculating it in the future) or not.

Details

The plotHeatmap function calls aheatmap to draw the heatmap. The main points of plotHeatmap are to 1) allow for different matrix inputs, separating out the color scale visualization and the clustering of the samples/features. 2) to visualize the clusters and meta data with the heatmap. The intended use case is to allow the user to visualize the original count scale of the data (on the log-scale), but create the hierarchical clustering on another, more appropriate dataset for clustering, such as normalized data. Similarly, some of the palettes in the package were developed assuming that the visualization might be on unscaled/uncentered data, rather than the residual from the mean of the gene, and thus palettes need to take on a greater range of relevant values so as to show meaningful comparisons with genes on very different scales.

If data is a ClusterExperiment object, visualizeData indicates what kind of transformation should be done to assay(data) for calculating the color scale. The features will be clustered based on these data as well. A different data.frame or matrix can be given for the visualization. For example, if the ClusterExperiment object contains normalized data, but the user wishes that the color scale be based on the log-counts for easier interpretation, visualizeData could be set to be the log2(counts + 1).

If data is a ClusterExperiment object, clusterSamplesData can be used to indicate the type of clustering for the samples. If equal to 'dendrogramValue' the dendrogram stored in data will be used; if dendrogram is missing, a new one will be created based on the primaryCluster of data using makeDendrogram, assuming no errors are created (if errors are created, then clusterSamplesData will be set to "primaryCluster"). If clusterSamplesData is equal to "hclust", then standard hierachical clustering of the transformed data will be used. If clusterSamplesData is equal to 'orderSamplesValue' no clustering of the samples will be done, and instead the samples will be ordered as in the slot orderSamples of data. If clusterSamplesData is equal to 'primaryCluster', again no clustering will be done, and instead the samples will be ordered based on grouping the samples to match the primaryCluster of data; however, if the primaryCluster of data is only one cluster or consists soley of -1/-2 values, clusterSamplesData will be set to "hclust". If clusterSamplesData is not a character value, clusterSamplesData can be a integer valued vector giving the order of the samples.

If data is a matrix, then colData is a data.frame of annotation data to be plotted above the heatmap and whColDataCont gives the index of the column(s) of this dataset that should be consider continuous. Otherwise the annotation data for colData will be forced into a factor (which will be nonsensical for continous data). If data is a ClusterExperiment object, colData should refer to a index or column name of the colData slot of data. In this case colData will be added to any choices of clusterings chosen by the whichClusters argument (if any). If both clusterings and sample data are chosen, the clusterings will be shown closest to data (i.e. on bottom).

If data is a ClusterExperiment object, clusterFeaturesData is not a dataset, but instead indicates which features should be shown in the heatmap. In this case clusterFeatures can be one of the following:

- "all" All rows/genes will be shown
- character giving dimensionality reductionShould match one of values saved in reducedDims slot or a builtin function in listBuiltInReducedDims(). nFeatures then gives the number of dimensions to show. The heatmap will then be of the dimension reduction vectors
- character giving filtering Should match one of values saved in filterStats slot or a builtin function in listBuiltInFilterStats(). nFeatures gives the number of genes to keep after filtering.
- character giving gene/row names
- vector of integers giving row indices

- a list of indices or rownamesThis is used to indicate that the features should be grouped according to the elements of the list, with blank (white) space between them (see makeBlankData for more details). In this case, no clustering is done of the features.

If breaks is a numeric value between 0 and 1, then breaks is assumed to indicate the upper quantile (on the log scale) at which the heatmap color scale should stop. For example, if breaks=0.9, then the breaks will evenly spaced up until the 0.9 upper quantile of data, and then all values after the 0.9 quantile will be absorbed by the upper-most color bin. This can help to reduce the visual impact of a few highly expressed genes (features).

Note that plotHeatmap calls aheatmap under the hood. This allows you to plot multiple heatmaps via par(mfrow=c(2,2)), etc. However, the dendrograms do not resize if you change the size of your plot window in an interactive session of R (this might be a problem for RStudio if you want to pop it out into a large window...). Also, plotting to a pdf adds a blank page; see help pages of aheatmap for how to turn this off.

clusterLegend takes the place of argument annColors from aheatmap for giving colors to the annotation on the heatmap. clusterLegend should be list of length equal to ncol(colData) with names equal to the colnames of colData. Each element of the list should be a either the format requested by aheatmap (a vector of colors with names corresponding to the levels of the column of colData), or should be format of the clusterLegend slot in a ClusterExperiment object. Color assignments to the rows/genes should also be passed via clusterLegend (assuming annRow is an argument passed to ...). If clusterFeaturesData is a *named* list describing groupings of genes then the colors for those groups can be given in clusterLegend under the name "Gene Group".

If you have a factor with many levels, it is important to note that aheatmap does not recycle colors across factors in the colData, and in fact runs out of colors and the remaining levels get the color white. Thus if you have many factors or many levels in those factors, you should set their colors via clusterLegend.

Many arguments can be passed on to aheatmap, however, some are set internally by plotHeatmap. In particular, setting the values of Rowv or Colv will cause errors. color in aheatmap is replaced by colorScale in plotHeatmap. The annCol to give annotation to the samples is replaced by the colData; moreover, the annColors option in aheatmap will also be set internally to give more vibrant colors than the default in aheatmap (for ClusterExperiment objects, these values can also be set in the clusterLegend slot). Other options should be passed on to aheatmap, though they have not been all tested. Useful options include treeheight=0 to suppress plotting of the dendrograms, annLegend=FALSE to suppress the legend of factors shown beside columns/rows, and cexRow=0 or cexCol=0 to suppress plotting of row/column labels.

plotCoClustering is a convenience function to plot the heatmap of the co-clustering distance matrix from the coClustering slot of a ClusterExperiment object (either by calculating the hamming distance of the clusterings stored in the coClustering slot, or the distance stored in the coClustering slot if it has already been calculated.

Value

Returns (invisibly) a list with elements

- aheatmapOut The output from the final call of aheatmap.
- colData the annotation data.frame given to the argument annCol in aheatmap.
- clusterLegend the annotation colors given to the argument annColors aheatmap.
- breaks The breaks used for aheatmap, after adjusting for quantile.

Author(s)

Elizabeth Purdom

See Also

aheatmap, makeBlankData, showHeatmapPalettes, makeDendrogram, dendrogram

Examples

```
data(simData)

cl <- rep(1:3,each=100)
cl2 <- cl
changeAssign <- sample(1:length(cl), 80)
cl2[changeAssign] <- sample(cl[changeAssign])
ce <- ClusterExperiment(simCount, cl2, transformation=function(x){log2(x+1)})

#simple, minimal, example. Show counts, but cluster on underlying means
plotHeatmap(ce)

#assign cluster colors
colors <- bigPalette[20:23]
names(colors) <- 1:3
plotHeatmap(data=simCount, clusterSamplesData=simData,
colData=data.frame(cl), clusterLegend=list(colors))

#show two different clusters
anno <- data.frame(cluster1=cl, cluster2=cl2)
out <- plotHeatmap(simData, colData=anno)

#return the values to see format for giving colors to the annotations
out$clusterLegend

#assign colors to the clusters based on plotClusters algorithm
plotHeatmap(simData, colData=anno, alignColData=TRUE)

#assign colors manually
annoColors <- list(cluster1=c("black", "red", "green"),
cluster2=c("blue","purple","yellow"))

plotHeatmap(simData, colData=anno, clusterLegend=annoColors)

#give a continuous valued -- need to indicate columns
anno2 <- cbind(anno, Cont=c(rnorm(100, 0), rnorm(100, 2), rnorm(100, 3)))
plotHeatmap(simData, colData=anno2, whColDataCont=3)

#compare changing breaks quantile on visual effect
## Not run:
par(mfrow=c(2,2))
plotHeatmap(simData, colorScale=seqPal1, breaks=1, main="Full length")
plotHeatmap(simData,colorScale=seqPal1, breaks=.99, main="0.99 Quantile Upper
Limit")
plotHeatmap(simData,colorScale=seqPal1, breaks=.95, main="0.95 Quantile Upper
Limit")
plotHeatmap(simData, colorScale=seqPal1, breaks=.90, main="0.90 Quantile
Upper Limit")

## End(Not run)
```

plotReducedDims *Plot 2-dimensionsal representation with clusters*

Description

Plot a 2-dimensional representation of the data, color-code by a clustering.

Usage

```
## S4 method for signature 'ClusterExperiment'
plotReducedDims(
  object,
  whichCluster = "primary",
  reducedDim = "PCA",
  whichDims = c(1, 2),
  plotUnassigned = TRUE,
  legend = TRUE,
  legendTitle = "",
  nColLegend = 6,
  clusterLegend = NULL,
  unassignedColor = NULL,
  missingColor = NULL,
  pch = 19,
  xlab = NULL,
  ylab = NULL,
  ...
)
```

Arguments

object	a ClusterExperiment object
whichCluster	argument that can be a single numeric or character value indicating the *single* clustering to be used. Giving values that result in more than one clustering will result in an error. See details of getClusterIndex.
reducedDim	What dimensionality reduction method to use. Should match either a value in reducedDimNames(object) or one of the built-in functions of listBuiltInReducedDims()
whichDims	vector of length 2 giving the indices of which dimensions to show. The first value goes on the x-axis and the second on the y-axis.
plotUnassigned	logical as to whether unassigned (either -1 or -2 cluster values) should be plotted.
legend	either logical, indicating whether to plot legend, or character giving the location of the legend (passed to legend)
legendTitle	character value giving title for the legend. If NULL, uses the clusterLabels value for clustering.
nColLegend	The number of columns in legend. If missing, picks number of columns internally.
clusterLegend	matrix with three columns and colnames 'clusterIds','name', and 'color' that give the color and name of the clusters in whichCluster. If NULL, pulls the information from object.

unassignedColor	If not NULL, should be character value giving the color for unassigned (-1) samples (overrides clusterLegend) default.
missingColor	If not NULL, should be character value giving the color for missing (-2) samples (overrides clusterLegend) default.
pch	the point type, passed to plot.default
xlab	Label for x axis
ylab	Label for y axis
...	arguments passed to plot.default

Details

If plotUnassigned=TRUE, and the color for -1 or -2 is set to "white", will be coerced to "lightgrey" regardless of user input to missingColor and unassignedColor. If plotUnassigned=FALSE, the samples with -1/-2 will not be plotted, nor will the category show up in the legend.

If the requested reducedDim method has not been created yet, the function will call makeReducedDims on the FIRST assay of x. The results of this method will be saved as part of the object and returned INVISIBLY (meaning if you don't save the output of the plotting command, the results will vanish). To pick another assay, you should call 'makeReducedDims' directly and specify the assay.

Value

A plot is created. Nothing is returned.

See Also

plot.default, makeReducedDims, listBuiltInReducedDims()

Examples

```
#clustering using pam: try using different dimensions of pca and different k
data(simData)

cl <- clusterMany(simData, nReducedDims=c(5, 10, 50), reducedDim="PCA",
clusterFunction="pam", ks=2:4, findBestK=c(TRUE,FALSE),
removeSil=c(TRUE,FALSE), makeMissingDiss=TRUE)

plotReducedDims(cl,legend="bottomright")
```

---

plottingFunctions *Convert clusterLegend into useful formats*

Description

Function for converting the information stored in the clusterLegend slot into other useful formats.

Most of these functions are called internally by plotting functions, but are exported in case the user finds them useful.

Usage

```
makeBlankData(
  data,
  groupsOfFeatures = NULL,
  groupsOfSamples = NULL,
  nBlankFeatures = 1,
  nBlankSamples = 1
)

## S4 method for signature 'ClusterExperiment'
convertClusterLegend(
  object,
  output = c("plotAndLegend", "aheatmapFormat", "matrixNames", "matrixColors"),
  whichClusters = ifelse(output == "plotAndLegend", "primary", "all")
)

showPalette(colPalette = bigPalette, which = NULL, cex = 1)

bigPalette

massivePalette

setBreaks(data, breaks = NA, makeSymmetric = FALSE, returnBreaks = TRUE)

showHeatmapPalettes()

seqPal5

seqPal2

seqPal3

seqPal4

seqPal1

## S4 method for signature 'ClusterExperiment'
plotClusterLegend(
  object,
  whichCluster = "primary",
  clusterNames,
  title,
  add = FALSE,
  location = if (add) "topright" else "center",
  ...
)
```

Arguments

data	matrix with samples on columns and features on rows.
groupsOfFeatures	list, with each element of the list containing a vector of numeric indices of fea-

	tures (rows).
groupsOfSamples	list, with each element of the list containing a vector of numeric indices of samples (columns).
nBlankFeatures	the number of blank lines to add in the data matrix to separate the groups of feature indices (will govern the amount of white space if data is then fed to heatmap.)
nBlankSamples	the number of blank lines to add in the data matrix to separate the groups of sample indices (will govern the amount of white space if data is then fed to heatmap.)
object	a ClusterExperiment object.
output	character value, indicating desired type of conversion.
whichClusters	argument that can be either numeric or character vector indicating the clusterings to be used. See details of getClusterIndex.
colPalette	a vector of character colors. By default, the palette bigPalette is used
which	numeric. Which colors to plot. Must be a numeric vector with values between 1 and length of colPalette. If missing, all colors plotted.
cex	numeric value giving the cex for the text of the plot.
breaks	either vector of breaks, or number of breaks (integer) or a number between 0 and 1 indicating a quantile, between which evenly spaced breaks should be calculated. If missing or NA, will determine evenly spaced breaks in the range of the data.
makeSymmetric	whether to make the range of the breaks symmetric around zero (only used if not all of the data is non-positive and not all of the data is non-negative)
returnBreaks	logical as to whether to return the vector of breaks. See details.
whichCluster	argument that can be a single numeric or character value indicating the *single* clustering to be used. Giving values that result in more than one clustering will result in an error. See details of getClusterIndex.
clusterNames	vector of names for the clusters; vector should have names that correspond to the clusterIds in the ClusterExperiment object. If this argument is missing, will use the names in the "name" column of the clusterLegend slot of the object.
title	title for the clusterLegend plot
add	logical. Whether legend should be added to the existing plot.
location	character passed to x argument of legend indicating where to place legend.
...	arguments passed to legend

Format

An object of class character of length 56.

An object of class character of length 484.

An object of class character of length 16.

An object of class character of length 14.

An object of class character of length 11.

An object of class character of length 13.

An object of class character of length 11.

Details

makeBlankData pulls the data corresponding to the row indices in groupsOfFeatures adds lines of NA values into data between these groups. When given to heatmap, will create white space between these groups of features.

convertClusterLegend pulls out information stored in the clusterLegend slot of the object and returns it in useful format.

bigPalette is a long palette of colors (length 58) used by plotClusters and accompanying functions. showPalette creates plot that gives index of each color in a vector of colors. massivePalette is a combination of bigPalette and the non-grey colors of colors() (length 487). massivePalette is mainly useful for when doing plotClusters of a very large number of clusterings, each with many clusters, so that the code doesn't run out of colors. However, many of the colors will be very similar to each other.

showPalette will plot the colPalette colors with their labels and index.

if returnBreaks if FALSE, instead of returning the vector of breaks, the function will just return the second smallest and second largest value of the breaks. This is useful for alternatively just setting values of the data matrix larger than these values to this value if breaks was a percentile. This argument is only used if breaks<1, indicating truncating the breaks for large values of data.

setBreaks gives a set of breaks (of length 52) equally spaced between the boundaries of the data. If breaks is between 0 and 1, then the evenly spaced breaks are between these quantiles of the data.

seqPal1-seqPal4 are palettes for the heatmap. showHeatmapPalettes will show you these palettes.

Value

makeBlankData returns a list with items

- "dataWBlanks" The data with the rows of NAs separating the given indices.
- "rowNamesWBlanks" A vector of characters giving the rownames for the data, including blanks for the NA rows. These are not given as rownames to the returned data because they are not necessarily unique. However, they can be given to the labRow argument of aheatmap or plotHeatmap.
- "colNamesWBlanks" A vector of characters giving the colnames for the data, including blanks for the NA rows. They can be given to the labCol argument of aheatmap or plotHeatmap.
- "featureGroupNamesWBlanks" A vector of characters of the same length as the number of rows of the new data (i.e. with blanks) giving the group name for the data, indicating which group (i.e. which element of groupsOfFeatures list) the feature came from. If groupsOfFeatures has unique names, these names will be used, other wise "Feature Group1", "Feature Group2", etc. The NA rows are given NA values.
- "sampleGroupNamesWBlanks" A vector of characters of the same length as the number of columns of the new data (i.e. with blanks) giving the group name for the data, indicating which group (i.e. which element of groupsOfFeatures list) the feature came from. If groupsOfFeatures has unique names, these names will be used, other wise "SampleGroup1", "Group2", etc. The NA rows are given NA values.

If output="plotAndLegend", "convertClusterLegend" will return a list that provides the necessary information to color samples according to cluster and create a legend for it:

- "colorVector" A vector the same length as the number of samples, assigning a color to each cluster of the primaryCluster of the object.
- "legendNames" A vector the length of the number of clusters of primaryCluster of the object giving the name of the cluster.

- "legendColors" A vector the length of the number of clusters of primaryCluster of the object giving the color of the cluster.

If `output="aheatmap"` a conversion of the clusterLegend to be in the format requested by `aheatmap`. The column 'name' is used for the names and the column 'color' for the color of the clusters.

If `output="matrixNames"` or `"matrixColors"` a matrix the same dimension of `clusterMatrix(object)`, but with the cluster color or cluster name instead of the clusterIds, respectively.

See Also

`plotHeatmap`

Examples

```
data(simData)

x <- makeBlankData(simData[,1:10], groupsOfFeatures=list(c(5, 2, 3), c(20,
34, 25)))
plotHeatmap(x$dataWBlanks,clusterFeatures=FALSE)
showPalette()
showPalette(massivePalette,cex=0.6)
setBreaks(data=simData,breaks=.9)

#show the palette colors
showHeatmapPalettes()

#compare the palettes on heatmap
cl <- clusterSingle(simData, subsample=FALSE,
sequential=FALSE,
mainClusterArgs=list(clusterFunction="pam", clusterArgs=list(k=8)))

## Not run:
par(mfrow=c(2,3))
plotHeatmap(cl, colorScale=seqPal1, main="seqPal1")
plotHeatmap(cl, colorScale=seqPal2, main="seqPal2")
plotHeatmap(cl, colorScale=seqPal3, main="seqPal3")
plotHeatmap(cl, colorScale=seqPal4, main="seqPal4")
plotHeatmap(cl, colorScale=seqPal5, main="seqPal5")
par(mfrow=c(1,1))

## End(Not run)
```

`renameClusters` *Change assigned names or colors of clusters*

Description

Change the assigned names or colors of the clusters in a clustering stored in the clusterLegend slot of the object.

Usage

```
## S4 method for signature 'ClusterExperiment,character'
renameClusters(
  object,
  value,
  whichCluster = "primary",
  matchTo = c("name", "clusterIds")
)

## S4 method for signature 'ClusterExperiment,character'
recolorClusters(
  object,
  value,
  whichCluster = "primary",
  matchTo = c("name", "clusterIds")
)
```

Arguments

object	a ClusterExperiment object.
value	The value to be substituted in the corresponding slot. See the slot descriptions in ClusterExperiment for details on what objects may be passed to these functions.
whichCluster	argument that can be a single numeric or character value indicating the *single* clustering to be used. Giving values that result in more than one clustering will result in an error. See details of getClusterIndex.
matchTo	whether to match to the cluster name ("name") or internal cluster id ("clusterIds")

Value

renameClusters changes the names assigned to clusters within a clustering

recolorClusters changes the colors assigned to clusters within a clustering

Examples

```
#create CE object
data(simData)
cl1 <- clusterSingle(simData, subsample=FALSE,
sequential=FALSE, mainClusterArgs=list(clusterArgs=list(k=3),
clusterFunction="pam"))
#Give names to the clusters
clusterLegend(cl1)
cl1<-renameClusters(cl1, c("1"="A","2"="B","3"="C"), matchTo="clusterIds")
clusterLegend(cl1)
# Change name of single one
cl1<-renameClusters(cl1, c("1"="D"), matchTo="clusterIds")
clusterLegend(cl1)
# Match to existing name, rather than clusterId
cl1<-renameClusters(cl1, c("B"="N"), matchTo="name")
clusterLegend(cl1)
# Change colors in similar way
cl1<-recolorClusters(cl1, c("N"="red"),matchTo=c("name"))
clusterLegend(cl1)
```

RSEC *Resampling-based Sequential Ensemble Clustering*

Description

Implementation of the RSEC algorithm (Resampling-based Sequential Ensemble Clustering) for single cell sequencing data. This is a wrapper function around the existing ClusterExperiment workflow that results in the output of RSEC.

Usage

```
## S4 method for signature 'SummarizedExperiment'
RSEC(x, ...)

## S4 method for signature 'data.frame'
RSEC(x, ...)

## S4 method for signature 'ClusterExperiment'
RSEC(x, eraseOld = FALSE, rerunClusterMany = FALSE, ...)

## S4 method for signature 'matrixOrHDF5'
RSEC(x, ...)

## S4 method for signature 'SingleCellExperiment'
RSEC(
  x,
  isCount = FALSE,
  transFun = NULL,
  reduceMethod = "PCA",
  nFilterDims = defaultNDims(x, reduceMethod, type = "filterStats"),
  nReducedDims = defaultNDims(x, reduceMethod, type = "reducedDims"),
  k0s = 4:15,
  subsample = TRUE,
  sequential = TRUE,
  clusterFunction = "hierarchical01",
  alphas = c(0.1, 0.2, 0.3),
  betas = 0.9,
  minSizes = 1,
  makeMissingDiss = if (ncol(x) < 1000) TRUE else FALSE,
  consensusProportion = 0.7,
  consensusMinSize,
  dendroReduce,
  dendroNDims,
  mergeMethod = "adjP",
  mergeCutoff,
  mergeLogFCcutoff,
  mergeDEMethod = if (isCount) "limma-voom" else "limma",
  verbose = FALSE,
  parameterWarnings = FALSE,
  mainClusterArgs = NULL,
  subsampleArgs = NULL,
```

```
  seqArgs = NULL,
  consensusArgs = NULL,
  whichAssay = 1,
  ncores = 1,
  random.seed = NULL,
  stopOnErrors = FALSE,
  run = TRUE
)
```

Arguments

x	the data matrix on which to run the clustering. Can be object of the following classes: matrix (with genes in rows), SummarizedExperiment, SingleCellExperiment or ClusterExperiment.
...	For signature matrix, arguments to be passed on to mclapply (if ncores>1). For all the other signatures, arguments to be passed to the method for signature matrix.
eraseOld	logical. Only relevant if input x is of class ClusterExperiment. If TRUE, will erase existing workflow results (clusterMany as well as mergeClusters and makeConsensus). If FALSE, existing workflow results will have "_i" added to the clusterTypes value, where i is one more than the largest such existing workflow clusterTypes.
rerunClusterMany	logical. If the object is a ClusterExperiment object, determines whether to rerun the clusterMany step. Useful if want to try different parameters for combining clusters after the clusterMany step, without the computational costs of the clusterMany step.
isCount	if transFun=NULL, then isCount=TRUE will determine the transformation as defined by function(x){log2(x+1)}, and isCount=FALSE will give a transformation function function(x){x}. Ignored if transFun=NULL. If object is of class ClusterExperiment, the stored transformation will be used and giving this parameter will result in an error.
transFun	a transformation function to be applied to the data. If the transformation applied to the data creates an error or NA values, then the function will throw an error. If object is of class ClusterExperiment, the stored transformation will be used and giving this parameter will result in an error.
reduceMethod	character A character identifying what type of dimensionality reduction to perform before clustering. Options are 1) "none", 2) one of listBuiltInReducedDims() or listBuiltInFitlerStats OR 3) stored filtering or reducedDim values in the object.
nFilterDims	vector of the number of the most variable features to keep (when "var", "abscv", or "mad" is identified in reduceMethod).
nReducedDims	vector of the number of dimensions to use (when reduceMethod gives a dimensionality reduction method).
k0s	the k0 parameter for sequential clustering (see seqCluster)
subsample	logical as to whether to subsample via subsampleClustering. If TRUE, clustering in mainClustering step is done on the co-occurance between clusterings in the subsampled clustering results. If FALSE, the mainClustering step will be run directly on x/diss

`sequential` logical whether to use the sequential strategy (see details of `seqCluster`). Can be used in combination with `subsample=TRUE` or `FALSE`.

`clusterFunction` function used for the clustering. This must be either 1) a character vector of built-in clustering techniques, or 2) a *named* list of `ClusterFunction` objects. Current functions can be found by typing `listBuiltInFunctions()` into the command-line.

`alphas` values of alpha to be tried. Only used for clusterFunctions of type '01'. Determines tightness required in creating clusters from the dissimilarity matrix. Takes on values in [0,1]. See documentation of `ClusterFunction`.

`betas` values of `beta` to be tried in sequential steps. Only used for `sequential=TRUE`. Determines the similarity between two clusters required in order to deem the cluster stable. Takes on values in [0,1]. See documentation of `seqCluster`.

`minSizes` the minimimum size required for a cluster (in the `mainClustering` step). Clusters smaller than this are not kept and samples are left unassigned.

`makeMissingDiss` logical. Whether to calculate necessary distance matrices needed when input is not "diss". If TRUE, then when a clustering function calls for a inputType "diss", but the given matrix is of type "X", the function will calculate a distance function. A dissimilarity matrix will also be calculated if a post-processing argument like `findBestK` or `removeSil` is chosen, since these rely on calcualting silhouette widths from distances.

`consensusProportion` passed to `proportion` in `makeConsensus`

`consensusMinSize` passed to `minSize` in `makeConsensus`

`dendroReduce` passed to `reduceMethod` in `makeDendrogram`

`dendroNDims` passed to `nDims` in `makeDendrogram`

`mergeMethod` passed to `mergeMethod` in `mergeClusters`

`mergeCutoff` passed to `cutoff` in `mergeClusters`

`mergeLogFCcutoff` passed to `logFCcutoff` in `mergeClusters`

`mergeDEMethod` passed to `DEMethod` argument in `mergeClusters`. By default, unless otherwise chosen by the user, if `isCount=TRUE`, then `mergeDEMethod="limma-voom"`, otherwise `mergeDEMethod="limma"`. These choices are for speed considerations and the user may want to try `mergeDEMethod="edgeR"` on smaller datasets of counts.

`verbose` logical. If TRUE it will print informative messages.

`parameterWarnings` logical, as to whether warnings and comments from checking the validity of the parameter combinations should be printed.

`mainClusterArgs` list of arguments to be passed for the mainClustering step, see help pages of `mainClustering`.

`subsampleArgs` list of arguments to be passed to the subsampling step (if `subsample=TRUE`), see help pages of `subsampleClustering`.

`seqArgs` list of arguments to be passed to `seqCluster`.

`consensusArgs` list of additional arguments to be passed to `makeConsensus`

whichAssay	numeric or character specifying which assay to use. See assay for details.
ncores	the number of threads
random.seed	a value to set seed before each run of clusterSingle (so that all of the runs are run on the same subsample of the data). Note, if 'random.seed' is set, argument 'ncores' should NOT be passed via subsampleArgs; instead set the argument 'ncores' of clusterMany directly (which is preferred for improving speed anyway).
stopOnErrors	logical. If FALSE, if RSEC hits an error *after* the clusterMany step, it will return the results up to that point, rather than generating a stop error. The text of error will be printed as a NOTE. This allows the user to get the results to that point, so as to not have to rerun the computationally heavy earlier steps. The TRUE option is only provided for debugging purposes.
run	logical. If FALSE, doesn't run clustering, but just returns matrix of parameters that will be run, for the purpose of inspection by user (with rownames equal to the names of the resulting column names of clMat object that would be returned if run=TRUE). Even if run=FALSE, however, the function will create the dimensionality reductions of the data indicated by the user input.

Details

Note that the argument isCount is mainly used when the input is a matrix or SingleCellExperiment Class and passed to clusterMany to set the transformation function of the data. However, if RSEC is being re-called on an existing ClusterExperiment object, it does not reset the transformation; in this case the only impact it will have is in setting the default value for DEMethod for mergeClusters step, but ONLY if mergeClusters hasn't already been calculated. To set arguments that allow you to recalculate the non-null probabilities of the hierarchy see mergeClusters.

Value

A ClusterExperiment object is returned containing all of the clusterings from the steps of RSEC

---

rsecFluidigm *Documentation of rsecFluidigm object*

Description

Documentation of the creation of rsecFluidigm, result of RSEC run on fluidigm data for vignette

Usage

```
makeRsecFluidigmObject(object)
```

Arguments

object	object given to functions

Format

rsecFluidigm is a ClusterExperiment object, the result of running RSEC on fluidigm data described in vignette and available in the scRNAseq package.

Details

The functions `makeRsecFluidigmObject` and `checkRsecFluidigmObject` are helper functions whose sole purpose is to create `rsecFluidigm` and check that the results are the same as expected. `makeRsecFluidigmObject` also serves as documentation of the specific RSEC call that was made to create the `rsecFluidigm` object, as well as filtering and normalization of the fluidigm data. The purpose of making them functions is internal, to help more easily mantain and check if changes to the package have affected the results.

Author(s)

Elizabeth Purdom <epurdom@stat.berkeley.edu>

See Also

`fluidigm`.

Examples

```
# see code used create rsecFluidigm
# (print out the function)
makeRsecFluidigmObject
#code actualy run to create rsecFluidigm:
## Not run:
library(clusterExperiment)
data(fluidigmData)
data(fluidigmColData)
se<-SummarizedExperiment(assays=fluidigmData, colData=fluidigmColData)
RNGversion("3.5.0")
rsecFluidigm<-makeRsecFluidigmObject(se)
# Internal function for checking got correct results...
clusterExperiment:::checkRsecFluidigmObject(rsecFluidigm)
usethis::use_data(rsecFluidigm,overwrite=FALSE)

## End(Not run)
```

---

search_pairs *Search pairs of samples that co-cluster across subsamples*

Description

Assume that our input is a matrix, with N columns and B rows (the number of subsamples), storing integers – the cluster labels.

Usage

```
search_pairs(clusterings)
```

Arguments

`clusterings`	a matrix with the cluster labels

Value

A matrix with the co-clusters, but only the lower triangle is populated.

`seqCluster` *Program for sequentially clustering, removing cluster, and starting again.*

Description

Given a data matrix, this function will call clustering routines, and sequentially remove best clusters, and iterate to find clusters.

Usage

```
seqCluster(
  inputMatrix,
  inputType,
  k0,
  subsample = TRUE,
  beta,
  top.can = 0.01,
  remain.n = 30,
  k.min = 3,
  k.max = k0 + 10,
  verbose = TRUE,
  subsampleArgs = NULL,
  mainClusterArgs = NULL,
  warnings = FALSE
)
```

Arguments

`inputMatrix`	numerical matrix on which to run the clustering or a `SummarizedExperiment`, `SingleCellExperiment`, or `ClusterExperiment` object.
`inputType`	a character vector defining what type of input is given in the `inputMatrix` argument. Must consist of values "diss","X", or "cat" (see details). "X" and "cat" should be indicate matrices with features in the row and samples in the column; "cat" corresponds to the features being numerical integers corresponding to categories, while "X" are continuous valued features. "diss" corresponds to an `inputMatrix` that is a NxN dissimilarity matrix. "cat" is largely used internally for clustering of sets of clusterings.
`k0`	the value of K at the first iteration of sequential algorithm, see details below or vignette.
`subsample`	logical as to whether to subsample via `subsampleClustering` to get the distance matrix at each iteration; otherwise the distance matrix is set by arguments to `mainClustering`.
`beta`	value between 0 and 1 to decide how stable clustership membership has to be before 'finding' and removing the cluster.
`top.can`	only the top.can clusters from `mainClustering` (ranked by 'orderBy' argument given to `mainClustering`) will be compared pairwise for stability. Can be either an integer value, identifying the absolute number of clusters, or a value between 0 and 1, meaning to keep all clusters with at least this proportion of

	the remaining samples in the cluster. Making this either a very big integer or equal to 0 will effectively remove this parameter and all pairwise comparisons of all clusters found will be considered; this might result in smaller clusters being found. If top.can is between 0 and 1, then there is still a hard threshold of at least 5 samples in a cluster to be considered as a cluster.
`remain.n`	when only this number of samples are left (i.e. not yet clustered) then algorithm will stop.
`k.min`	each iteration of sequential detection of clustering will decrease the beginning K of subsampling, but not lower than k.min.
`k.max`	algorithm will stop if K in iteration is increased beyond this point.
`verbose`	whether the algorithm should print out information as to its progress.
`subsampleArgs`	list of arguments to be passed to `subsampleClustering`.
`mainClusterArgs`	list of arguments to be passed to `mainClustering`).
`warnings`	logical. Whether to print out the many possible warnings and messages regarding checking the internal consistency of the parameters.

Details

`seqCluster` is not meant to be called by the user. It is only an exported function so as to be able to clearly document the arguments for `seqCluster` which can be passed via the argument `seqArgs` in functions like `clusterSingle` and `clusterMany`.

This code is adapted from the sequential protion of the code of the tightClust package of Tseng and Wong. At each iteration of the algorithm it finds a set of samples that constitute a homogeneous cluster and remove them, and iterate again to find the next set of samples that form a cluster.

In each iteration, to determine the next set of homogeneous set of samples, the algorithm will iteratively cluster the current set of samples for a series of increasing values of the parameter K, starting at a value `kinit` and increasing by 1 at each iteration, until a sufficiently homogeneous set of clusters is found. For the first set of homogeneous samples, `kinit` is set to the argument $k0$, and for iteration, `kinit` is increased internally.

Depending on the value of `subsample` how the value of K is used differs. If `subsample=TRUE`, K is the `k` sent to the cluster function `clusterFunction` sent to `subsampleClustering` via `subsampleArgs`; then `mainClustering` is run on the result of the co-occurance matrix from `subsampleClustering` with the `ClusterFunction` object defined in the argument `clusterFunction` set via `mainClusterArgs`. The number of clusters actually resulting from this run of `mainClustering` may not be equal to the K sent to the clustering done in `subsampleClustering`. If `subsample=FALSE`, `mainClustering` is called directly on the data to determine the clusters and K set by `seqCluster` for this iteration determines the parameter of the clustering done by `mainClustering`. Specifically, the argument `clusterFunction` defines the clustering of the `mainClustering` step and `k` is sent to that `ClusterFunction` object. This means that if `subsample=FALSE`, the `clusterFunction` must be of `algorithmType` "K".

In either setting of `subsample`, the resulting clusters from `mainClustering` for a particular K will be compared to clusters found in the previous iteration of $K-1$. For computational (and other?) convenience, only the first `top.can` clusters of each iteration will be compared to the first `top.can` clusters of previous iteration for similarity (where `top.can` currently refers to ordering by size, so first `top.can` largest clusters.

If there is no cluster of the first `top.can` in the current iteration K that has overlap similarity > `beta` to any in the previous iteration, then the algorithm will move to the next iteration, increasing to $K+1$.

If, however, of these clusters there is a cluster in the current iteration K that has overlap similarity > beta to a cluster in the previous iteration $K-1$, then the cluster with the largest such similarity will be identified as a homogenous set of samples and the samples in it will be removed and designated as such. The algorithm will then start again to determine the next set of homogenous samples, but without these samples. Furthermore, in this case (i.e. a cluster was found and removed), the value of `kinit` will be be reset to `kinit-1`; i.e. the range of increasing K that will be iterated over to find a set of homogenous samples will start off one value less than was the case for the previous set of homogeneous samples. If `kinit-1<k.min`, then `kinit` will be set to `k.min`.

If there are less than `remain.n` samples left after finding a cluster and removing its samples, the algorithm will stop, as subsampling is deamed to no longer be appropriate. If the K has to be increased to beyond `k.max` without finding any pair of clusters with overlap > beta, then the algorithm will stop. Any samples not found as part of a homogenous set of clusters at that point will be classified as unclustered (given a value of -1)

Certain combinations of inputs to `mainClusterArgs` and `subsampleArgs` are not allowed. See `clusterSingle` for these explanations.

Value

A list with values

- `clustering` a vector of length equal to nrows(x) giving the integer-valued cluster ids for each sample. The integer values are assigned in the order that the clusters were found. "-1" indicates the sample was not clustered.
- `clusterInfo` if clusters were successfully found, a matrix of information regarding the algorithm behavior for each cluster (the starting and stopping K for each cluster, and the number of iterations for each cluster).
- `whyStop` a character string explaining what triggered the algorithm to stop.

References

Tseng and Wong (2005), "Tight Clustering: A Resampling-Based Approach for Identifying Stable and Tight Patterns in Data", Biometrics, 61:10-16.

See Also

tight.clust, `clusterSingle`,`mainClustering`,`subsampleClustering`

Examples

```
## Not run:
data(simData)

set.seed(12908)
clustSeqHier <- seqCluster(simData, inputType="X", k0=5, subsample=TRUE,
   beta=0.8, subsampleArgs=list(resamp.n=100,
   samp.p=0.7, clusterFunction="kmeans", clusterArgs=list(nstart=10)),
   mainClusterArgs=list(minSize=5,clusterFunction="hierarchical01",
   clusterArgs=list(alpha=0.1)))

## End(Not run)
```

simData *Simulated data for running examples*

Description

Simulated data for running examples

Format

Three objects are loaded, two data frame(s) of simulated data each with 300 samples/columns and 153 variables/rows, and a vector of length 300 with the true cluster assignments.

Details

`simData` is simulated normal data of 300 observations with 51 relevant variables and the rest of the variables being noise, with observations being in one of 3 groups. `simCount` is simulated count data of the same dimensions. `trueCluster` gives the true cluster identifications of the samples. The true clusters are each of size 100 and are in order in the columns of the data.frames.

Author(s)

Elizabeth Purdom <epurdom@stat.berkeley.edu>

Examples

```
#code used to create data:
## Not run:
nvar<-51 #multiple of 3
n<-100
x<-cbind(matrix(rnorm(n*nvar,mean=5),nrow=nvar),
 matrix(rnorm(n*nvar,mean=-5),nrow=nvar),
          matrix(rnorm(n*nvar,mean=0),nrow=nvar))
#make some of them flipped effects (better for testing if both sig under/over
#expressed variables)
geneGroup<-sample(rep(1:3,each=floor(nvar/3)))
gpIndex<-list(1:n,(n+1):(n*2),(2*n+1):(n*3))
x[geneGroup==1,]<-x[geneGroup==1,unlist(gpIndex[c(3,1,2)])]
x[geneGroup==2,]<-x[geneGroup==2,unlist(gpIndex[c(2,3,1)])]

#add in differences in variable means
smp<-sample(1:nrow(x),10)
x[smp,]<-x[smp,]+10

#make different signal y
y<-cbind(matrix(rnorm(n*nvar,mean=1),nrow=nvar),
         matrix(rnorm(n*nvar,mean=-1),nrow=nvar),
         matrix(rnorm(n*nvar,mean=0),nrow=nvar))
y<-y[,sample(1:ncol(y))]+ matrix(rnorm(3*n*nvar,sd=3),nrow=nvar)

#add together the two signals
simData<-x+y

#add pure noise variables
simData<-rbind(simData,matrix(rnorm(3*n*nvar,mean=10),nrow=nvar),
```

```
                matrix(rnorm(3*n*nvar,mean=5),nrow=nvar))
#make count data
countMean<-exp(simData/2)
simCount<-matrix(rpois(n=length(as.vector(countMean)), lambda
=as.vector(countMean)+.1),nrow=nrow(countMean),ncol=ncol(countMean))
#labels for the truth
trueCluster<-rep(c(1:3),each=n)
save(list=c("simCount","simData","trueCluster"),file="data/simData.rda")

## End(Not run)
```

`subsampleClustering` *Cluster subsamples of the data*

Description

Given input data, this function will subsample the samples, cluster the subsamples, and return a n x n matrix with the probability of co-occurance.

Usage

```
## S4 method for signature 'character'
subsampleClustering(clusterFunction, ...)

## S4 method for signature 'ClusterFunction'
subsampleClustering(
  clusterFunction,
  inputMatrix,
  inputType,
  clusterArgs = NULL,
  classifyMethod = c("All", "InSample", "OutOfSample"),
  resamp.num = 100,
  samp.p = 0.7,
  ncores = 1,
  warnings = TRUE,
  ...
)
```

Arguments

`clusterFunction`	a `ClusterFunction` object that defines the clustering routine. See `ClusterFunction` for required format of user-defined clustering routines. User can also give a character value to the argument `clusterFunction` to indicate the use of clustering routines provided in package. Type `listBuiltInFunctions` at command prompt to see the built-in clustering routines. If `clusterFunction` is missing, the default is set to "pam".
`...`	arguments passed to mclapply (if ncores>1).
`inputMatrix`	numerical matrix on which to run the clustering or a `SummarizedExperiment`, `SingleCellExperiment`, or `ClusterExperiment` object.

inputType	a character vector defining what type of input is given in the inputMatrix argument. Must consist of values "diss","X", or "cat" (see details). "X" and "cat" should be indicate matrices with features in the row and samples in the column; "cat" corresponds to the features being numerical integers corresponding to categories, while "X" are continuous valued features. "diss" corresponds to an inputMatrix that is a NxN dissimilarity matrix. "cat" is largely used internally for clustering of sets of clusterings.
clusterArgs	a list of parameter arguments to be passed to the function defined in the clusterFunction slot of the ClusterFunction object. For any given ClusterFunction object, use function requiredArgs to get a list of required arguments for the object.
classifyMethod	method for determining which samples should be used in calculating the co-occurance matrix. "All"= all samples, "OutOfSample"= those not subsampled, and "InSample"=those in the subsample. See details for explanation.
resamp.num	the number of subsamples to draw.
samp.p	the proportion of samples to sample for each subsample.
ncores	integer giving the number of cores. If ncores>1, mclapply will be called.
warnings	logical as to whether should give warning if arguments given that don't match clustering choices given. Otherwise, inapplicable arguments will be ignored without warning.

Details

subsampleClustering is not usually called directly by the user. It is only an exported function so as to be able to clearly document the arguments for subsampleClustering which can be passed via the argument subsampleArgs in functions like clusterSingle and clusterMany.

requiredArgs: The choice of "All" or "OutOfSample" for requiredArgs require the classification of arbitrary samples not originally in the clustering to clusters; this is done via the classifyFUN provided in the ClusterFunction object. If the ClusterFunction object does not have such a function to define how to classify into a cluster samples not in the subsample that created the clustering then classifyMethod must be "InSample". Note that if "All" is chosen, all samples will be classified into clusters via the classifyFUN, not just those that are out-of-sample; this could result in different assignments to clusters for the in-sample samples than their original assignment by the clustering depending on the classification function. If you do not choose 'All',it is possible to get NAs in resulting S matrix (particularly if when not enough subsamples are taken) which can cause errors if you then pass the resulting D=1-S matrix to mainClustering. For this reason the default is "All".

Value

A n x n matrix of co-occurances, i.e. a symmetric matrix with [i,j] entries equal to the percentage of subsamples where the ith and jth sample were clustered into the same cluster. The percentage is only out of those subsamples where the ith and jth samples were both assigned to a clustering. If classifyMethod=="All", this is all subsamples for all i,j pairs. But if classifyMethod=="InSample" or classifyMethod=="OutOfSample", then the percentage is only taken on those subsamples where the ith and jth sample were both in or out of sample, respectively, relative to the subsample.

Examples

```
## Not run:
#takes a bit of time, not run on checks:
```

```
data(simData)
coOccur <- subsampleClustering( inputMatrix=simData, inputType="X",
clusterFunction="kmeans",
clusterArgs=list(k=3,nstart=10), resamp.n=100, samp.p=0.7)

#visualize the resulting co-occurance matrix
plotHeatmap(coOccur)

## End(Not run)
```

---

subset *Functions to subset ClusterExperiment Objects*

Description

These functions are used to subset ClusterExperiment objects, either by removing samples, genes, or clusterings

Usage

```
## S4 method for signature 'ClusterExperiment'
removeClusterings(x, whichClusters)

## S4 method for signature 'ClusterExperiment'
removeClusters(
  x,
  whichCluster,
  clustersToRemove,
  makePrimary = FALSE,
  clusterLabels = NULL
)

## S4 method for signature 'ClusterExperiment,ANY,character,ANY'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'ClusterExperiment,ANY,logical,ANY'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'ClusterExperiment,ANY,numeric,ANY'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'ClusterExperiment'
subsetByCluster(
  x,
  clusterValue,
  whichCluster = "primary",
  matchTo = c("name", "clusterIds")
)
```

Arguments

x	a ClusterExperiment object.
whichClusters	argument that can be either numeric or character vector indicating the clusterings to be used. See details of getClusterIndex.
whichCluster	argument that can be a single numeric or character value indicating the *single* clustering to be used. Giving values that result in more than one clustering will result in an error. See details of getClusterIndex.
clustersToRemove	numeric vector identifying the clusters to remove (whose samples will be reassigned to -1 value).
makePrimary	whether to make the added cluster the primary cluster (only relevant if y is a vector)
clusterLabels	label(s) for the clusters being added. If y a matrix, the column names of that matrix will be used by default, if clusterLabels is not given.
i, j	A vector of logical or integer subscripts, indicating the rows and columns to be subsetted for i and j, respectively.
...	The arguments transformation, clusterTypes and clusterInfo to be passed to the constructor for signature SingleCellExperiment,matrix.
drop	A logical scalar that is ignored.
clusterValue	values of the cluster to match to for subsetting
matchTo	whether to match to the cluster name ("name") or internal cluster id ("clusterIds")

Details

removeClusterings removes the clusters given by whichClusters. If the primaryCluster is one of the clusters removed, the primaryClusterIndex is set to 1 and the dendrogram and coclustering matrix are discarded and orderSamples is set to 1:NCOL(x).

removeClusters creates a new cluster that unassigns samples in cluster clustersToRemove (in the clustering defined by whichClusters) and assigns them to -1 (unassigned)

Note that when subsetting the data, the dendrogram information and the co-clustering matrix are lost.

Value

A ClusterExperiment object.

removeClusterings returns a ClusterExperiment object, unless all clusters are removed, in which case it returns a SingleCellExperiment object.

subsetByCluster subsets the object by clusters in a clustering and returns a ClusterExperiment object with only those samples

Examples

```
#load CE object
data(rsecFluidigm)
# remove the mergeClusters step from the object
clusterLabels(rsecFluidigm)
test<-removeClusterings(rsecFluidigm,whichClusters="mergeClusters")
clusterLabels(test)
tableClusters(rsecFluidigm)
```

```
test<-removeClusters(rsecFluidigm,whichCluster="mergeClusters",clustersToRemove=c("m01","m04"))
tableClusters(test,whichCluster="mergeClusters")
```

`transformData` *Transform the original data in a ClusterExperiment object*

Description

Provides the transformed data

Usage

```
## S4 method for signature 'matrixOrHDF5'
transformData(object, transFun = NULL, isCount = FALSE)

## S4 method for signature 'ClusterExperiment'
transformData(object, whichAssay = 1, ...)

## S4 method for signature 'SingleCellExperiment'
transformData(object, whichAssay = 1, ...)

## S4 method for signature 'SummarizedExperiment'
transformData(object, ...)
```

Arguments

`object`	a matrix, SummarizedExperiment, SingleCellExperiment or ClusterExperiment object.
`transFun`	a transformation function to be applied to the data. If the transformation applied to the data creates an error or NA values, then the function will throw an error. If object is of class `ClusterExperiment`, the stored transformation will be used and giving this parameter will result in an error.
`isCount`	if `transFun=NULL`, then `isCount=TRUE` will determine the transformation as defined by `function(x){log2(x+1)}`, and `isCount=FALSE` will give a transformation function `function(x){x}`. Ignored if `transFun=NULL`. If object is of class `ClusterExperiment`, the stored transformation will be used and giving this parameter will result in an error.
`whichAssay`	numeric or character specifying which assay to use. See `assay` for details.
`...`	Values passed on the the 'matrix' method.

Details

The data matrix defined by `assay(x)` is transformed based on the transformation function either defined in x (in the case of a `ClusterExperiment` object) or by user given values for other classes.

Value

A DataFrame defined by `assay(x)` suitably transformed

Examples

```
mat <- matrix(data=rnorm(200), ncol=10)
mat[1,1] <- -1 #force a negative value
labels <- gl(5, 2)
cc <- ClusterExperiment(mat, as.numeric(labels), transformation =
function(x){x^2}) #define transformation as x^2
z<-transformData(cc)
```

`updateObject` *Update old ClusterExperiment object to current class definition*

Description

This function updates ClusterExperiment objects from previous versions of package into the current definition

Usage

```
## S4 method for signature 'ClusterExperiment'
updateObject(object, checkTransformAndAssay = FALSE, ..., verbose = FALSE)
```

Arguments

`object`	a `ClusterExperiment` (or `clusterExperiment` from older versions). Must have at a minimum a slot `clusterMatrix`.
`checkTransformAndAssay`	logical. Whether to check the content of the assay and given transformation function for whether they are valid.
`...`	Additional arguments, for use in specific `updateObject` methods.
`verbose`	`TRUE` or `FALSE`, indicating whether information about the update should be reported. Use `message` to report this information.

Details

The function creates a valid `ClusterExperiment` object by adding the default values of missing slots. It does so by calling the `ClusterExperiment` function, which imputs default (empty) values for missing slots.

The object is required to have minimal components to be updated. Specifically, it must have all the required elements of a Summarized Experiment as well as the basic slots of a ClusterExperiment object which have not changed over time. These are: `clusterMatrix`, `primaryIndex`, `clusterInfo`, `transformation`, `clusterTypes`, `clusterLegend`, `orderSamples`.

If *any* of the dendrogram-related slots are missing, ALL of the dendrogram *and* merge related slots will be cleared to default values. Similarly, if *any* of the merge-related slots are missing, ALL of the merge-related slots will be cleared to the default values.

Cluster and Sample dendrograms of the class `dendrogram` will be updated to the `phylo4d` class now used in `ClusterExperiment` objects; the merge information on these nodes will be updated to have the correct format (i.e. match to the internal node id names in the new dendrogram). The previous identification of nodes that was previously created internally by plotDendrogram and the merging (labels in the form of 'Node1','Node2'), will be kept as `nodeLabels` in the new dendrogram class.

The function currently only works for object of `ClusterExperiment`, not the older name `clusterExperiment`.

Value

A valid ClusterExperiment object based on the current definition of ClusterExperiment.

See Also

ClusterExperiment

---

workflowClusters *Methods for workflow clusters*

Description

The main workflow of the package is made of clusterMany, makeConsensus, and mergeClusters. The clusterings from these functions (and not those obtained in a different way) can be obtained with the functions documented here.

Usage

```
## S4 method for signature 'ClusterExperiment'
workflowClusters(x, iteration = 0)

## S4 method for signature 'ClusterExperiment'
workflowClusterDetails(x)

## S4 method for signature 'ClusterExperiment'
workflowClusterTable(x)

## S4 method for signature 'ClusterExperiment'
setToCurrent(x, whichCluster, eraseOld = FALSE)

## S4 method for signature 'ClusterExperiment'
setToFinal(x, whichCluster, clusterLabel)
```

Arguments

x	a ClusterExperiment object.
iteration	numeric. Which iteration of the workflow should be used.
whichCluster	argument that can be a single numeric or character value indicating the *single* clustering to be used. Giving values that result in more than one clustering will result in an error. See details of getClusterIndex.
eraseOld	logical. Only relevant if input x is of class ClusterExperiment. If TRUE, will erase existing workflow results (clusterMany as well as mergeClusters and makeConsensus). If FALSE, existing workflow results will have "_i" added to the clusterTypes value, where i is one more than the largest such existing workflow clusterTypes.
clusterLabel	optional string value to give to cluster set to be "final"

Value

`workflowClusters` returns a matrix consisting of the appropriate columns of the `clusterMatrix` slot.

`workflowClusterDetails` returns a `data.frame` with some details on the clusterings, such as the type (e.g., 'clusterMany', 'makeConsensus') and iteration.

`workflowClusterTable` returns a table of how many of the clusterings belong to each of the following possible values: 'final', 'mergeClusters', 'makeConsensus' and 'clusterMany'.

`setToCurrent` returns a `ClusterExperiment` object where the indicated cluster of `whichCluster` has been set to the most current iteration in the workflow. Pre-existing clusters are appropriately updated.

`setToFinal` returns a `ClusterExperiment` object where the indicated cluster of `whichCluster` has clusterType set to "final". The primaryClusterIndex is also set to this cluster, and the clusterLabel, if given.

Examples

```
data(simData)

cl <- clusterMany(simData,nReducedDims=c(5,10,50),  reduceMethod="PCA",
clusterFunction="pam", ks=2:4, findBestK=c(FALSE), removeSil=TRUE,
subsample=FALSE, makeMissingDiss=TRUE)

clCommon <- makeConsensus(cl, whichClusters="workflow", proportion=0.7,
minSize=10)

clCommon <- makeDendrogram(clCommon)

clMerged <- mergeClusters(clCommon,mergeMethod="adjP", DEMethod="limma")

head(workflowClusters(clMerged))
workflowClusterDetails(clMerged)
workflowClusterTable(clMerged)
```

Index