Package 'clusterExperiment'

April 11, 2018

Title Compare Clusterings for Single-Cell Sequencing

Version 1.4.0

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

Author Elizabeth Purdom [aut, cre, cph], Davide Risso [aut], Marla Johnson [ctb]

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

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

License Artistic-2.0

Depends R (>= 3.4), SummarizedExperiment

Imports methods, NMF, RColorBrewer, ape, phylobase, cluster, stats, limma, dendextend, howmany, locfdr, matrixStats, graphics, parallel, RSpectra, kernlab, stringr

Suggests BiocStyle, knitr, testthat, scRNAseq, MAST

VignetteBuilder knitr

LazyData false

RoxygenNote 6.0.1

biocViews Clustering, RNASeq, Sequencing, Software, SingleCell

Collate 'AllChecks.R' 'AllClasses.R' 'AllGenerics.R' 'AllHelper.R' 'AllHelperClusterFunction.R' 'JiashinJiCode.R' 'addClusters.R' 'internalClusterFunctions.R' 'internalFunctions.R' 'builtInClusterFunctions.R' 'clusterContrasts.R' 'clusterLabels.R' 'clusterMany.R' 'clusterSingle.R' 'combineMany.R' 'dataCreation.R' 'getFeatures.R' 'getParams.R' 'mainClustering.R' 'makeBlankData.R' 'makeDendrogram.R' 'mergeClusters.R' 'plotBarplot.R' 'plotClusters.R' 'plotClustersWorkflow.R' 'plotContrastHeatmap.R' 'plotDendrogram.R' 'plotDimReduce.R' 'plotHeatmap.R' 'plottingHelpers.R' 'rsec.R' 'seqCluster.R' 'subsampleClustering.R' 'transformFunction.R' 'workflowClusters.R'

NeedsCompilation no

R topics documented:

`addClusters,ClusterExperiment,matrix-method`
Functions to add/remove clusters to ClusterExperiment

Description

These functions are used to add or remove clusters to a `ClusterExperiment` object.

Usage

```
## S4 method for signature 'ClusterExperiment,matrix'
addClusters(x, y, clusterTypes = "User")

## S4 method for signature 'ClusterExperiment,ClusterExperiment'
addClusters(x, y)

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

```
addClusters(x, y, clusterLabel = NULL,
  ...)

## S4 method for signature 'ClusterExperiment,character'
removeClusters(x, whichRemove,
  exactMatch = TRUE)

## S4 method for signature 'ClusterExperiment,numeric'
removeClusters(x, whichRemove)

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

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', 'combineMany' are reserved for the clustering coming from the package workflow and should not be used when creating a new object with the constructor.
clusterLabel	label(s) for the clusters being added.
...	Passed to signature `ClusterExperiment,matrix`.
whichRemove	which clusters to remove. Can be numeric or character. If numeric, must give indices of `clusterMatrix(x)` to remove. If character, should match a `clusterTypes` of x.
exactMatch	logical. Whether `whichRemove` must exactly match a value of `clusterTypes(x)`. Only relevant if whichRemove is character.

Details

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

`removeClusters` removes the clusters given by `whichRemove`. If all clusters are implied, then returns a `SummarizedExperiment` object. If the `primaryCluster` is one of the clusters removed, the `primaryClusterIndex` is set to 1 and the dendrogram and cooccurance matrix are discarded and orderSamples is set to `1:NCOL(x)`.

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

Value

A `ClusterExperiment` object with the added clusters.

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"))

addClusters(cl1, cl2)
```

`clusterContrasts,ClusterExperiment-method`

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"), removeNegative = 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.
`removeNegative`	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 removeNegative=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,nPCADims=c(5,10,50),  dimReduce="PCA",
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")
```

`ClusterExperiment-class`

Class ClusterExperiment

Description

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

In addition to the slots of the SummarizedExperiment 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, combineMany, ...) 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(se, clusters, ...)

## S4 method for signature 'matrix,ANY'
clusterExperiment(se, clusters, ...)

## S4 method for signature 'SummarizedExperiment,numeric'
clusterExperiment(se, clusters, ...)

## S4 method for signature 'SummarizedExperiment,character'
clusterExperiment(se, clusters, ...)

## S4 method for signature 'SummarizedExperiment,factor'
clusterExperiment(se, clusters, ...)

## S4 method for signature 'SummarizedExperiment,matrix'
clusterExperiment(se, clusters,
  transformation, primaryIndex = 1, clusterTypes = "User",
  clusterInfo = NULL, orderSamples = 1:ncol(se), dendro_samples = NULL,
  dendro_index = NA_real_, dendro_clusters = NULL, dendro_outbranch = NA,
  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_,
  checkTransformAndAssay = TRUE)
```

Arguments

se	a matrix or SummarizedExperiment containing the data to be 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 SummarizedExperiment,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', 'combineMany' 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` dendrogram. Sets the 'dendro_samples' slot (see Slots).

`dendro_index` numeric. Sets the `dendro_index` slot (see Slots).

`dendro_clusters` dendrogram. Sets the 'dendro_clusters' slot (see Slots).

`dendro_outbranch` logical. Sets the `dendro_outbranch` 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)

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

Details

The `clusterExperiment` constructor function gives clusterLabels based on the column names of the input matrix/SummarizedExperiment. 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.

`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

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

`merge_method` character indicating method used for merging

- `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.

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

`dendro_samples` dendrogram. A dendrogram containing the cluster relationship (leaves are samples; see `makeDendrogram` for details).

`dendro_clusters` dendrogram. A dendrogram containing the cluster relationship (leaves are clusters; see `makeDendrogram` 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.

`dendro_outbranch` logical. Whether the dendro_samples dendrogram put missing/non-clustered samples in an outbranch, or intermixed in the dendrogram.

`coClustering` matrix. A 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.

Examples

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

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

`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,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'
show(object)

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

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

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

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

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

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

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

## S4 method for signature 'ClusterExperiment,missing'
clusterMatrix(x, whichClusters)

## S4 method for signature 'ClusterExperiment,numeric'
clusterMatrix(x, whichClusters)

## S4 method for signature 'ClusterExperiment,character'
clusterMatrix(x, whichClusters)

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

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

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

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

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

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

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

## S4 method for signature 'ClusterExperiment'
clusterInfo(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,character'
tableClusters(x, whichClusters, ...)

## S4 method for signature 'ClusterExperiment,missing'
tableClusters(x, whichClusters, ...)

## S4 method for signature 'ClusterExperiment,numeric'
tableClusters(x, whichClusters, ...)
```

Arguments

`x, object`	a ClusterExperiment object.
`..., i, j, drop`	Forwarded to the `SummarizedExperiment` method.
`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 func-

tions.

whichClusters — optional argument that can be either numeric or character value. If numeric, gives the indices of the clusterMatrix to return; this can also be used to defined an ordering for the clusterings. whichClusters can be a character value identifying the clusterTypes to be used, or if not matching clusterTypes then clusterLabels; alternatively whichClusters can be either 'all' or 'workflow' to indicate choosing all clusters or choosing all workflowClusters. If missing, the entire matrix of all clusterings is returned.

Details

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

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

clusterMatrixNamed returns a matrix with cluster labels.

primaryClusterNamed returns the primary cluster (using cluster labels).

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

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

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

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

clusterMatrix returns the matrix with all the clusterings.

clusterMatrix returns the matrix with all the clusterings.

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).

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.

clusterInfo 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.

`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 '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 'ClusterFunction'
inputType(object)

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

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

Arguments

`object`	input to the method, usually either a `ClusterFunction` class or a character describing a built-in `ClusterFunction` object.
`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,matrix-method

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 'matrix'
clusterMany(x, dimReduce = "none", nVarDims = NA,
  nPCADims = NA, transFun = NULL, isCount = FALSE, ...)

## S4 method for signature 'list'
clusterMany(x, ks = NA, clusterFunction, alphas = 0.1,
  findBestK = FALSE, sequential = FALSE, removeSil = FALSE,
  subsample = FALSE, silCutoff = 0, distFunction = NA, betas = 0.9,
  minSizes = 1, verbose = FALSE, mainClusterArgs = NULL,
  subsampleArgs = NULL, seqArgs = NULL, ncores = 1, random.seed = NULL,
  run = TRUE, ...)

## S4 method for signature 'ClusterExperiment'
clusterMany(x, dimReduce = "none",
  nVarDims = NA, nPCADims = NA, eraseOld = FALSE, ...)

## S4 method for signature 'SummarizedExperiment'
clusterMany(x, dimReduce = "none",
  nVarDims = NA, nPCADims = NA, transFun = NULL, isCount = FALSE, ...)

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

Arguments

`x`	the data matrix on which to run the clustering. Can be: matrix (with genes in rows), a list of datasets overwhich the clusterings should be run, a `SummarizedExperiment` object, or a `ClusterExperiment` object.

dimReduce — character A character identifying what type of dimensionality reduction to perform before clustering. Options are "none","PCA", "var","cv", and "mad". See transform for more details.

nVarDims — vector of the number of the most variable features to keep (when "var", "cv", or "mad" is identified in dimReduce). If NA is included, then the full dataset will also be included.

nPCADims — vector of the number of PCs to use (when 'PCA' is identified in dimReduce). If NA is included, then the full dataset will also be included.

transFun — function A function to use to transform the input data matrix before clustering.

isCount — logical. Whether the data are in counts, in which case the default transFun argument is set as log2(x+1). This is simply a convenience to the user, and can be overridden by giving an explicit function to transFun.

... — For signature list, arguments to be passed on to mclapply (if ncores>1). For all the other signatures, arguments to be passed to the method for signature list.

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. Note that unlike in clusterSingle, this must be a character vector of pre-defined clustering techniques, and can not be a user-defined function. 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.

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.
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.
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 combineMany). 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.

Value

If run=TRUE and the input is either a matrix, a SummarizedExperiment object, or a ClusterExperiment object, 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=TRUE and the input is a list of data sets, a list with the following objects:

- clMat a matrix with each column corresponding to a clustering and each row to a sample.
- clusterInfo a list with information regarding clustering results (only relevant entries for those clusterings with sequential=TRUE)
- 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

If run=FALSE a list similar to that described above, but without the clustering results.

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,nPCADims=c(5,10,50),  dimReduce="PCA",
clusterFunction="pam",
ks=2:4,findBestK=c(TRUE,FALSE),removeSil=c(TRUE,FALSE),run=FALSE)
print(head(checkParams$paramMatrix))

#Now actually run it
cl <- clusterMany(simData,nPCADims=c(5,10,50),  dimReduce="PCA",
clusterFunction="pam",ks=2:4,findBestK=c(TRUE,FALSE),removeSil=c(TRUE,FALSE))
print(cl)
head(colnames(clusterMatrix(cl)))

#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))

par(mar=c(2, 10, 1, 1))
plotClusters(clMat, axisLine=-2)


## 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",
```

```
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, SummarizedExperiment, or ClusterExperiment object, this function will find clusters, based on a single specification of parameters.

Usage

```
## S4 method for signature 'missing,matrixOrNULL'
clusterSingle(x, diss, ...)

## S4 method for signature 'matrixOrNULL,missing'
clusterSingle(x, diss, ...)

## S4 method for signature 'SummarizedExperiment,missing'
clusterSingle(x, diss, ...)

## S4 method for signature 'ClusterExperiment,missing'
clusterSingle(x,
  replaceCoClustering = FALSE, ...)

## S4 method for signature 'matrixOrNULL,matrixOrNULL'
clusterSingle(x, diss, subsample = TRUE,
  sequential = FALSE, mainClusterArgs = NULL, subsampleArgs = NULL,
  seqArgs = NULL, isCount = FALSE, transFun = NULL,
  dimReduce = c("none", "PCA", "var", "cv", "mad"), ndims = NA,
  clusterLabel = "clusterSingle", checkDiss = TRUE)
```

Arguments

x	the data on which to run the clustering (features in rows), or a SummarizedExperiment, or ClusterExperiment object.
diss	n x n data matrix of dissimilarities between the samples on which to run the clustering.
...	arguments to be passed on to the method for signature matrix.
replaceCoClustering	logical. Applicable if x is a ClusterExperiment object. If TRUE, the co-clustering resulting from subsampling is returned in the coClustering object and replaces any existing coClustering object in the slot coClustering.
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.
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	logical. Whether the data are in counts, in which case the default transFun argument is set as log2(x+1). This is simply a convenience to the user, and can be overridden by giving an explicit function to transFun.
transFun	function A function to use to transform the input data matrix before clustering.
dimReduce	character A character identifying what type of dimensionality reduction to perform before clustering. Options are "none","PCA", "var","cv", and "mad". See transform for more details.
ndims	integer An integer identifying how many dimensions to reduce to in the reduction specified by dimReduce
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.
checkDiss	logical. Whether to check whether the input diss is valid.

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 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 input was x a matrix (or assay of a ClusterExperiment or SummarizedExperiment object).

If input was diss, 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
- diss: The dissimilarity matrix used in the clustering

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 ...)
   clusterFunction="hierarchical01",clusterArgs=list(alpha=0.1)))

## End(Not run)

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

`combineMany,matrix,missing-method`

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,missing'
combineMany(x, whichClusters, proportion,
  clusterFunction = "hierarchical01", propUnassigned = 0.5, minSize = 5)

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

## S4 method for signature 'ClusterExperiment,character'
combineMany(x, whichClusters, ...)

## S4 method for signature 'ClusterExperiment,missing'
combineMany(x, whichClusters, ...)
```

Arguments

`x`	a matrix or `clusterExperiment` object.
`whichClusters`	a numeric or character vector that specifies which clusters to compare (missing if x is a matrix)
`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 to use (passed to `mainClustering`); currently must be of type '01'.

`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)

`minSize`
: minimum size required for a set of samples to be considered in a cluster because of shared clustering, passed to `mainClustering`

`eraseOld`
: logical. Only relevant if input `x` is of class `ClusterExperiment`. If TRUE, will erase existing workflow results (clusterMany as well as mergeClusters and combineMany). 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 "combineMany", to indicate that this clustering is the result of a call to combineMany. However, a more informative label can be set (see vignette).

`...`
: arguments to be passed on to the method for signature `matrix,missing`.

Details

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 `combineMany` and the `percentageShared` matrix stored in the `coClustering` slot.

Examples

```
data(simData)

cl <- clusterMany(simData,nPCADims=c(5,10,50),  dimReduce="PCA",
clusterFunction="pam", ks=2:4, findBestK=c(FALSE), removeSil=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 <- combineMany(clMat, proportion=1, minSize=10)

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

oldpar <- par()
par(mar=c(1.1, 12.1, 1.1, 1.1))
plotClusters(cbind("70%Similarity"=clCommon70$clustering, clMat,
"100%Similarity"=clCommon100$clustering), axisLine=-2)

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

`getBestFeatures,matrix-method`

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 'matrix'
getBestFeatures(x, cluster, contrastType = c("F", "Dendro",
  "Pairs", "OneAgainstAll"), dendro = NULL, pairMat = NULL,
  contrastAdj = c("All", "PerContrast", "AfterF"), isCount = FALSE,
  normalize.method = "none", ...)

## S4 method for signature 'ClusterExperiment'
getBestFeatures(x, contrastType = c("F",
  "Dendro", "Pairs", "OneAgainstAll"), isCount = FALSE, ...)
```

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`.

`contrastAdj` What type of FDR correction to do for contrasts tests (i.e. if contrastType='Dendro' or 'Pairs').

`isCount` logical as to whether input data is count data, in which case to perform voom correction to data. See details.

`normalize.method` character value, passed to `voom` in `limma` package. Only used if `countData=TRUE`. Note that the default value is set to "none", which is not the default value of `voom`.

`...` options to pass to `topTable` or `topTableF` (see `limma` package)

Details

getBestFeatures returns the top ranked features corresponding to a cluster assignment. It uses limma to fit the models, and limma's 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)

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.

`isCount` triggers whether the "voom" correction will be performed in `limma`. If the input data is a matrix is counts (or a 'ClusterExperiment' object with counts as the primary data before transformation) this should be set to TRUE and they will be log-transformed internally by voom for the differential expression analysis in a way that accounts for the difference in the mean-variance relationships. Otherwise, dat should be on the correct (log) scale for differential expression analysis without a need a variance stabilization (e.g. microarray data). Currently the default is set to FALSE,

simply because the isCount has not been heavily tested. If the But TRUE with x being counts really should be the default for RNA-Seq data. If the input data is a 'ClusterExperiment' object, setting 'isCount=TRUE' will cause the program to ignore the internally stored transformation function and instead use voom with log2(x+0.5). Alternatively, `isCount=FALSE` for a `ClusterExperiment` object will cause the DE to be performed with `limma` after transforming the data with the stored transformation. Although some writing about "voom" seem to suggest that it would be appropriate for arbitrary transformations, the authors have cautioned against using it for anything other than count data on mailing lists. For this reason we are not implementing it for arbitrary transformations at this time (e.g. log(FPKM+epsilon) transformations).

Value

A `data.frame` in the same format as `topTable`, except for the following additional or changed columns:

- `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.

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

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),
isCount=FALSE)

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

#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)

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

# do DE on counts using voom
# compare results to if used simData instead (not on count scale).
# Again, not relevant for this silly example, but basic principle useful
testFVoom <- getBestFeatures(simCount, primaryCluster(cl), contrastType="F",
number=nrow(simData), isCount=TRUE)
plot(testF$P.Value[order(testF$Index)],
testFVoom$P.Value[order(testFVoom$Index)],log="xy")
```

`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")
```

Arguments

`x`	a ClusterExperiment object that contains clusterings from running `clusterMany`.
`whichClusters`	The indices (or clusterLabels) of those clusters whose labels will be parsed to determine the parameters; should be subset of the clusterMany results.

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.

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, nPCADims=c(5, 10, 50), dimReduce="PCA",
clusterFunction="pam", ks=2:4, findBestK=c(TRUE,FALSE),
removeSil=c(TRUE,FALSE))
getClusterManyParams(cl)
```

---

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 bassed 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.

Required arguments for clusterFUN:

- "x or diss"either x and/or diss depending on inputType. If x, then x is assumed to be nfeatures x nsamples (like assay(CEObj) would give)
- "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!); 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 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).

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

A ClusterFunction object.

Slots

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

inputType a character defining what type of input clusterFUN takes. Must be one of either "diss","X", or "either"

`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 (see details below).

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

`inputClassifyType` the input type for the classification function (if not NULL); like `inputType`, must be one of "diss","X", or "either"

`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).

`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(x,diss,k,checkArgs,cluster.only,...){
cluster::pam(x=t(x),k=k,cluster.only=cluster.only)
}
#passes internal check
internalFunctionCheck(goodFUN,inputType="X",algorithmType="K",outputType="vector")
#Note it doesn't pass if inputType="either" because no catches for x=NULL
internalFunctionCheck(goodFUN, inputType="either",algorithmType="K",outputType="vector")
myCF<-clusterFunction(clusterFUN=goodFUN, inputType="X",algorithmType="K", outputType="vector")
badFUN<-function(x,diss,k,checkArgs,cluster.only,...){cluster::pam(x=x,k=k)}
internalFunctionCheck(badFUN,inputType="X",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 is "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"
- "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, x = NULL,
  diss = NULL, distFunction = NA, clusterArgs = NULL, minSize = 1,
  orderBy = c("size", "best"), format = c("vector", "list"),
```

```
  checkArgs = TRUE, checkDiss = TRUE, returnData = FALSE, ...)

## 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.
x	p x n data matrix on which to run the clustering (samples in columns).
diss	n x n data matrix of dissimilarities between the samples on which to run the clustering
distFunction	a distance function to be applied to D. Only relevant if input is only x (a matrix of data), and diss=NULL. See details of clusterSingle for the required format of the distance function.
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.
checkArgs	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.
checkDiss	logical. Whether to check whether the input diss is valid.
returnData	logical as to whether to return the diss or x matrix in the output. If FALSE only the clustering vector is returned.

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

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.

Examples

```
data(simData)
cl1<-mainClustering(x=simData,clusterFunction="pam",clusterArgs=list(k=3))
cl2<-mainClustering(simData,clusterFunction="hierarchical01",clusterArgs=list(alpha=.1))
cl3<-mainClustering(simData,clusterFunction="tight",clusterArgs=list(alpha=.1))
#change distance to manhattan distance
cl4<-mainClustering(simData,clusterFunction="pam",clusterArgs=list(k=3),
     distFunction=function(x){dist(x,method="manhattan")})

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

#do tight
clustSubTight <- mainClustering(simData, clusterFunction="tight", clusterArgs=list(alpha=0.1),
minSize=5)

#two twists to pam
clustSubPamK <- mainClustering(simData, clusterFunction="pam", silCutoff=0, minSize=5,
removeSil=TRUE, clusterArgs=list(k=3))
clustSubPamBestK <- mainClustering(simData, clusterFunction="pam", silCutoff=0,
minSize=5, 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 checkArgs=FALSE)
clustSubTight_test <- mainClustering(simData, clusterFunction="tight",
clusterArgs=list(alpha=0.1), minSize=5, removeSil=TRUE)
clustSubTight_test2 <- mainClustering(simData, clusterFunction="tight",
clusterArgs=list(alpha=0.1,evalClusterMethod="average"))
```

`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", dimReduce = c("mad", "cv", "var", "PCA",
  "none"), ndims = if (dimReduce %in% c("mad", "cv", "var")) 500 else if
  (dimReduce == "none") NA else 50, ignoreUnassignedVar = TRUE,
  unassignedSamples = c("outgroup", "cluster"), ...)

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

Arguments

x	data to define the medoids from. Matrix and ClusterExperiment supported.
whichCluster	an integer index or character string that identifies which cluster should be used to make the dendrogram. Default is primaryCluster.
dimReduce	character A character identifying what type of dimensionality reduction to perform before clustering. Options are "none","PCA", "var","cv", and "mad". See transform for more details.
ndims	integer An integer identifying how many dimensions to reduce to in the reduction specified by dimReduce
ignoreUnassignedVar	logical indicating whether dimensionality reduction via top feature variability (i.e. 'var','cv','mad') should ignore unassigned samples in the primary clustering for calculation of the top features.
unassignedSamples	how to handle unassigned samples("-1") ; only relevant for sample clustering. See 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.

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 mergeDendrogram.

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.

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,matrix-method

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 'matrix'
mergeClusters(x, cl, dendro = NULL,
  mergeMethod = c("none", "Storey", "PC", "adjP", "locfdr", "MB", "JC"),
  plotInfo = c("none", "all", "Storey", "PC", "adjP", "locfdr", "MB", "JC",
  "mergeMethod"), nodePropTable = NULL, calculateAll = TRUE,
  showWarnings = FALSE, cutoff = 0.1, plot = TRUE, isCount = TRUE, ...)

## S4 method for signature 'ClusterExperiment'
mergeClusters(x, eraseOld = FALSE,
  isCount = FALSE, mergeMethod = "none", plotInfo = "all",
  clusterLabel = "mergeClusters", leafType = c("samples", "clusters"),
  plotType = c("colorblock", "name", "ids"), plot = TRUE, ...)

## 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'
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 merging is plotted at each node. If 'none', then no proportions will be added to the dendrogram. 'plotInfo' can also be one of the mergeMethod choices (even if that method is not the method chosen in 'mergeMethod' options).
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

isCount	logical as to whether input data is a count matrix. See details.
...	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 combineMany). 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).
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.

Count correction If isCount=TRUE, and the input is a matrix, log2(count + 1) will be used for makeDendrogram and the original data with voom correction will be used in getBestFeatures). If input is ClusterExperiment, then setting isCount=TRUE also means that the log2(1+count) will be used as the transformation, like for the matrix case as well as the voom calculation, and will NOT use the transformation stored in the object. If FALSE, then transform(x) will be given to the input and will be used for both makeDendrogram and getBestFeatures, with no voom correction.

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.

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.
- propDE A table of the proportions that are DE on each node.
- originalClusterDendro 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.

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.

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)

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

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

plotBarplot,ClusterExperiment,character-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,character'
plotBarplot(object, whichClusters, ...)

## S4 method for signature 'ClusterExperiment,missing'
plotBarplot(object, whichClusters, ...)

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

## S4 method for signature 'ClusterExperiment,missing'
plotBarplot(object, whichClusters, ...)

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

## S4 method for signature 'matrix,missing'
plotBarplot(object, whichClusters, xNames = NULL,
  legNames = NULL, legend = TRUE, xlab = NULL, legend.title = NULL,
  unassignedColor = "white", missingColor = "grey",
  colPalette = bigPalette, ...)
```

Arguments

object	A matrix of with each column corresponding to a clustering and each row a sample or a ClusterExperiment object.
whichClusters	If numeric, a predefined order for the clusterings in the plot. If x is a ClusterExperiment object, whichClusters can be a character value identifying the clusterTypes to be used, or if not matching clusterTypes then clusterLabels; alternatively whichClusters can be either 'all' or 'workflow' to indicate choosing all clusters or choosing all workflowClusters.
...	for plotBarplot arguments passed either to the method of plotBarplot for matrices or ultimately to barplot.
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").
xNames	names for the first clusters (on x-axis). By default uses values in 1st cluster of clusters matrix

legNames	names for the first clusters (in legend). By default uses values in 2nd cluster of clusters matrix
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. Must be as long as the maximum number of clusters found in any single clustering/column given in object or will otherwise return an error.

Details

The first column of the cluster matrix will be on the x-axis and the second column 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, combineMany, clusterMany.

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, nPCADims=c(5, 10, 50), dimReduce="PCA",
clusterFunction="pam", ks=2:4, findBestK=c(TRUE,FALSE),
removeSil=c(TRUE,FALSE))

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

`plotClusters,ClusterExperiment,character-method`
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,character'
plotClusters(object,
  whichClusters = c("workflow", "all"), ...)

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

## S4 method for signature 'ClusterExperiment,missing'
plotClusters(object, whichClusters, ...)

## S4 method for signature 'matrix,missing'
plotClusters(object, whichClusters,
  orderSamples = NULL, sampleData = NULL, reuseColors = FALSE,
  matchToTop = FALSE, plot = TRUE, unassignedColor = "white",
  missingColor = "grey", minRequireColor = 0.3, startNewColors = FALSE,
  colPalette = bigPalette, 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`	If numeric, a predefined order for the clusterings in the plot. If x is a `ClusterExperiment` object, `whichClusters` can be a character value identifying the `clusterTypes` to be used, or if not matching `clusterTypes` then `clusterLabels`; alternatively `whichClusters` can be either 'all' or 'workflow' to indicate choosing all clusters or choosing all `workflowClusters`.
`...`	for `plotClusters` arguments passed either to the method of `plotClusters` for matrices, or ultimately to `plot` (if `add=FALSE`).
`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 give new colors for the remaining (not implemented yet). '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.

`sampleData` If `clusters` is a matrix, `sampleData` gives a matrix of additional cluster/sampleData on the samples to be plotted with the clusterings given in clusters. Values in `sampleData` will be added to the end (bottom) of the plot. NAs in the `sampleData` matrix will trigger an error. If `clusters` is a `ClusterExperiment` object, the input in `sampleData` refers to columns of the the `colData` slot of the `ClusterExperiment` object to be plotted with the clusters. In this case, `sampleData` 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 `sampleData` 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.

`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 (some of the colors at the end of bigPalette are a bit wonky, and so if you have a large clusters matrix, this can be useful).

`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, combineMany, 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.

- `index` a vector of length equal to `ncols(clusters)` giving the order of the columns 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[index,]` 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 matrix is 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.

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, nPCADims=c(5, 10, 50), dimReduce="PCA",
clusterFunction="pam", ks=2:4, findBestK=c(TRUE,FALSE),
removeSil=c(TRUE,FALSE))

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
showBigPalette(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)
```

---

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", "combineMany"),
  whichClusterMany = NULL, nBlankLines = ceiling(nClusters(object) * 0.05),
  existingColors = FALSE, nSizeResult = ceiling(nClusters(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 clusterMany results. Can be numeric or character vector, indicating the indices or clusterLabels/clusterTypes of the clusterings of interest, respectively.
whichClusterMany	numeric indices of which of the clusterMany clusterings to plot (if NULL, defaults to all). Unlike whichClusters, these must be numeric indices. They must also refer to clusterings of clusterType clusterMany.
nBlankLines	the number of blank (i.e. white) rows to add between the clusterMany clusterings and the highlighted clusterings.
existingColors	logical. If logical, whether the highlighted clusters should use colors matched to the clusterMany results, or should the stored colors in ClusterExperiment object be used. This argument has no effect on the colors of the clusterMany results, whose colors will be chosen based on the alignment of plotClusters.
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 combineMany 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, nPCADims=c(5, 10, 50), dimReduce="PCA",
clusterFunction="pam", ks=2:4, findBestK=c(TRUE,FALSE),
removeSil=c(TRUE,FALSE))
cl <- combineMany(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, ...)
```

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 names in clusterLegend(object).
...	Arguments passed to plotHeatmap

Value

A heatmap is created. Nothing 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, isCount=FALSE)
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("clusters", "samples"), plotType = c("name", "colorblock",
  "ids"), mergeInfo = c("none", "all", "Storey", "PC", "adjP", "locfdr", "MB",
  "JC", "mergeMethod"), main, sub, clusterLabelAngle = 45,
  removeOutbranch = TRUE, legend = "side", ...)
```

Arguments

x	a ClusterExperiment object.
whichClusters	only used if leafType="samples"). If numeric, an index for the clusterings to be plotted with dendrogram. Otherwise, whichClusters can be a character value identifying the clusterTypes to be used, or if not matching clusterTypes then clusterLabels; alternatively whichClusters can be either 'all' or 'workflow' or 'primaryCluster' to indicate choosing all clusters or choosing all workflowClusters. Default 'dendro' indicates using the clustering that created the dendrogram.
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.
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).
...	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. Nothing is returned.

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",cutoff=0.1,plot=FALSE)
plotDendrogram(cl)
plotDendrogram(cl,leafType="samples",whichClusters="all",plotType="colorblock")
```

`plotDimReduce,ClusterExperiment,character-method`
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,character'
plotDimReduce(object, whichClusters,
  ...)

## S4 method for signature 'ClusterExperiment,missing'
plotDimReduce(object, whichClusters, ...)

## S4 method for signature 'ClusterExperiment,numeric'
plotDimReduce(object, whichClusters,
  dimReduce = c("PCA"), whichDims = c(1:2), plotUnassigned = TRUE,
  legend = TRUE, legendTitle = "", clusterLegend = NULL,
  unassignedColor = NULL, missingColor = NULL, pch = 19, xlab = NULL,
  ylab = NULL, ...)
```

Arguments

`object`	a ClusterExperiment object
`whichClusters`	which clusters to show on the plot
`...`	arguments passed to `plot.default`
`dimReduce`	What dimensionality reduction method to use (currently only PCA). See `transform` for how this is implemented.
`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. If TRUE, and the color for -1 is set to "white", will be coerced to "lightgrey". To change this, set a different (non-white) color in `unassignedColor` argument.
`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.

clusterLegend	matrix with three columns and colnames 'clusterIds','name', and 'color' that give the color and name of the clusters in whichClusters. If NULL, pulls the information from object.
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.
pch	the point type, passed to plot.default
xlab	Label for x axis
ylab	Label for y axis

Value

A plot is created. Nothing is returned.

See Also

plot.default, transform

Examples

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

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

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

---

plotHeatmap,SummarizedExperiment-method

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 'SummarizedExperiment'
plotHeatmap(data, isCount = FALSE,
  transFun = NULL, ...)

## S4 method for signature 'ClusterExperiment'
plotHeatmap(data,
  clusterSamplesData = c("dendrogramValue", "hclust", "orderSamplesValue",
  "primaryCluster"), clusterFeaturesData = c("var", "all", "PCA"),
```

```
  nFeatures = NULL, visualizeData = c("transformed", "centeredAndScaled",
  "original"), whichClusters = c("primary", "workflow", "all", "none"),
  sampleData = NULL, clusterFeatures = TRUE, nBlankLines = 2, colorScale,
  ...)

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

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

## S4 method for signature 'matrix'
plotHeatmap(data, sampleData = NULL,
  clusterSamplesData = NULL, clusterFeaturesData = NULL,
  whSampleDataCont = NULL, clusterSamples = TRUE, showSampleNames = FALSE,
  clusterFeatures = TRUE, showFeatureNames = FALSE, colorScale = seqPal5,
  clusterLegend = NULL, alignSampleData = FALSE,
  unassignedColor = "white", missingColor = "grey", breaks = NA,
  isSymmetric = FALSE, overRideClusterLimit = FALSE, plot = TRUE, ...)

## S4 method for signature 'ClusterExperiment'
plotCoClustering(data,
  invert = ifelse(!is.null(data@coClustering) && all(diag(data@coClustering)
  == 0), TRUE, FALSE), ...)
```

Arguments

data	data to use to determine the heatmap. Can be a matrix, ClusterExperiment or SummarizedExperiment object. The interpretation of parameters depends on the type of the input to data.
isCount	logical. Whether the data are in counts, in which case the default transFun argument is set as log2(x+1). This is simply a convenience to the user, and can be overridden by giving an explicit function to transFun.
transFun	function A function to use to transform the input data matrix before clustering.
...	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 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 character string, or vector of characters or integers, indicating what clusters should be visualized with the heatmap.

sampleData If input to data is either a ClusterExperiment or SummarizedExperiment object, then sampleData must index the sampleData 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, sampleData is a matrix of additional data on the samples to show above heatmap. In this case, unless indicated by whSampleDataCont, sampleData 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.

whSampleDataCont
Which of the sampleData columns are continuous and should not be converted to counts. NULL indicates no additional sampleData. 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, sampleData columns will be assigned colors internally. clusterLegend should be list of length equal to ncol(sampleData) with names equal to the colnames of sampleData. 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 sampleData), or should be format of ClusterExperiment. This takes the place of argument annColors from aheatmap. 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 (see details) then the colors for those groups can be given in clusterLegend under the name "Gene Group".

alignSampleData
Logical as to whether should align the colors of the sampleData (only if clusterLegend not given and sampleData 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

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.

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.)

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 sampleData is a data.frame of annotation data to be plotted above the heatmap and whSampleDataCont gives the index of the column(s) of this dataset that should be con-

sider continuous. Otherwise the annotation data for sampleData will be forced into a factor (which will be nonsensical for continous data). If data is a ClusterExperiment object, sampleData should refer to a index or column name of the colData slot of data. In this case sampleData 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. "var" selects the nFeatures most variable genes (based on transformation(assay(data))); "PCA" results in a heatmap of the top nFeatures PCAs of the transformation(assay(data)). clusterFeaturesData can also be a vector of characters or integers, indicating the rownames or indices respectively of assay(data) that should be shown. For all of these options, the features are clustered based on the visualizeData data. Finally, in the ClusterExperiment version of plotHeatmap, clusterFeaturesData can be a list of indices or rownames, indicating 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.

If you have a factor with many levels, it is important to note that aheatmap does not recycle colors across factors in the sampleData, 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 sampleData; 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.

plotCoClustering is a convenience function to plot the heatmap of the co-clustering matrix stored in the coClustering slot of a ClusterExperiment object.

Value

Returns (invisibly) a list with elements

- aheatmapOut The output from the final call of aheatmap.
- sampleData 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

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,
sampleData=data.frame(cl), clusterLegend=list(colors))

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

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

plotHeatmap(simData, sampleData=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, sampleData=anno2, whSampleDataCont=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)
```

`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, nBlankLines = 1)

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

showBigPalette(wh = NULL)

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

bigPalette

showHeatmapPalettes()

seqPal5

seqPal2

seqPal3

seqPal4

seqPal1
```

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.
`nBlankLines`	the number of blank lines to add in the data matrix to separate the groups of 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.

wh	numeric. Which colors to plot. Must be a numeric vector with values between 1 and 62.
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.
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)

Format

An object of class character of length 60.

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 62) used by plotClusters and accompanying functions. showBigPalette creates plot that gives index of each color in bigPalette.

showBigPalette will plot the bigPalette functions with their labels and index.

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.
- "groupNamesWBlanks" 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 "Group1", "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)
showBigPalette()
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)
```

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 'matrix'
RSEC(x, isCount = FALSE, transFun = NULL,
  dimReduce = "PCA", nVarDims = NA, nPCADims = c(50), k0s = 4:15,
  clusterFunction = "hierarchical01", alphas = c(0.1, 0.2, 0.3),
```

```
  betas = 0.9, minSizes = 1, combineProportion = 0.7,
  combineMinSize = 5, dendroReduce = "mad", dendroNDims = 1000,
  mergeMethod = "adjP", mergeCutoff = 0.05, verbose = FALSE,
  mainClusterArgs = NULL, subsampleArgs = NULL, seqArgs = NULL,
  ncores = 1, random.seed = NULL, run = TRUE)

## 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, ...)
```

Arguments

x	the data matrix on which to run the clustering. Can be: matrix (with genes in rows), a list of datasets overwhich the clusterings should be run, a SummarizedExperiment object, or a ClusterExperiment object.
isCount	logical. Whether the data are in counts, in which case the default transFun argument is set as log2(x+1). This is simply a convenience to the user, and can be overridden by giving an explicit function to transFun.
transFun	function A function to use to transform the input data matrix before clustering.
dimReduce	character A character identifying what type of dimensionality reduction to perform before clustering. Options are "none","PCA", "var","cv", and "mad". See transform for more details.
nVarDims	vector of the number of the most variable features to keep (when "var", "cv", or "mad" is identified in dimReduce). If NA is included, then the full dataset will also be included.
nPCADims	vector of the number of PCs to use (when 'PCA' is identified in dimReduce). If NA is included, then the full dataset will also be included.
k0s	the k0 parameter for sequential clustering (see seqCluster)
clusterFunction	function used for the clustering. Note that unlike in clusterSingle, this must be a character vector of pre-defined clustering techniques, and can not be a user-defined function. 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.
combineProportion	passed to proportion in combineMany

combineMinSize	passed to minSize in combineMany
dendroReduce	passed to dimReduce in makeDendrogram
dendroNDims	passed to ndims in makeDendrogram
mergeMethod	passed to mergeMethod in mergeClusters
mergeCutoff	passed to cutoff in mergeClusters
verbose	logical. If TRUE it will print informative messages.
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.
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.
...	For signature list, arguments to be passed on to mclapply (if ncores>1). For all the other signatures, arguments to be passed to the method for signature list.
eraseOld	logical. Only relevant if input x is of class ClusterExperiment. If TRUE, will erase existing workflow results (clusterMany as well as mergeClusters and combineMany). 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.

Value

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

---

rsecFluidigm *RSEC run for vignette*

Description

RSEC run for vignette

Format

clusterExperiment object, the result of running RSEC on fluidigm data described in vignette and available in the scRNAseq package.

Author(s)

Elizabeth Purdom <epurdom@stat.berkeley.edu>

See Also

fluidigm

Examples

```
#code used to create rsecFluidigm:
## Not run:
library(scRNAseq)
data("fluidigm")
se <- fluidigm[,colData(fluidigm)[,"Coverage_Type"]=="High"]
wh_zero <- which(rowSums(assay(se))==0)
pass_filter <- apply(assay(se), 1, function(x) length(x[x >= 10]) >= 10)
se <- se[pass_filter,]
fq <- round(limma::normalizeQuantiles(assay(se)))
assays(se) <- list(normalized_counts=fq)
wh<-which(colnames(colData(se)) %in% c("Cluster1","Cluster2"))
colnames(colData(se))[wh]<-c("Published1","Published2")
library(clusterExperiment)
ncores<-1
system.time(rsecFluidigm<-RSEC(se, isCount = TRUE,nPCADims=10,
ncores=ncores,random.seed=176201, clusterFunction="hierarchical01",
 combineMinSize=3))
packageVersion("clusterExperiment")
devtools::use_data(rsecFluidigm)

## End(Not run)
```

---

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(x = NULL, diss = NULL, k0, subsample = TRUE, beta,
  top.can = 5, remain.n = 30, k.min = 3, k.max = k0 + 10,
  verbose = TRUE, subsampleArgs = NULL, mainClusterArgs = NULL,
  checkDiss = TRUE)
```

Arguments

x	p x n data matrix on which to run the clustering (samples in columns).
diss	n x n data matrix of dissimilarities between the samples on which to run the clustering
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. Making this very big will effectively remove this parameter and all pairwise comparisons of all clusters found will be considered. This might result in smaller clusters being found. The current default is fairly large, so probably will have little effect.
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).
checkDiss	logical. Whether to check whether the input diss is valid.

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, 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, x = NULL,
  diss = NULL, distFunction = NA, clusterArgs = NULL,
  classifyMethod = c("All", "InSample", "OutOfSample"), resamp.num = 100,
  samp.p = 0.7, ncores = 1, checkArgs = TRUE, checkDiss = TRUE,
  largeDataset = FALSE, ...)
```

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).
x	the data on which to run the clustering (samples in columns).
diss	a dissimilarity matrix on which to run the clustering.
distFunction	a distance function to be applied to D. Only relevant if input is only x (a matrix of data), and diss=NULL. See details of clusterSingle for the required format of the distance function.
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.
checkArgs	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.
checkDiss	logical. Whether to check whether the input diss is valid.
largeDataset	logical indicating whether a more memory-efficient version should be used because the dataset is large. This is a beta option, and is in the process of being tested before it becomes the default.

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(clusterFunction="kmeans", x=simData,
clusterArgs=list(k=3,nstart=10), resamp.n=100, samp.p=0.7)

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

## End(Not run)
```

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

Description

Provides the transformed data (as defined by the object), as well as dimensionality reduction.

Usage

```
## S4 method for signature 'ClusterExperiment'
transform(`_data`, nPCADims = NA,
  nVarDims = NA, dimReduce = "none", ignoreUnassignedVar = FALSE)
```

Arguments

`_data`	a ClusterExperiment object.
`nPCADims`	Numeric vector giving the number of PC dimensions to use in PCA dimensionality reduction. If NA no PCA dimensionality reduction is done. nPCADims can also take values between (0,1) to indicate keeping the number of PCA dimensions necessary to account for that proportion of the variance.
`nVarDims`	Numeric (integer) vector giving the number of features (e.g. genes) to keep, based on variance/cv/mad variability.
`dimReduce`	Character vector specifying the dimensionality reduction to perform, any combination of 'none', 'PCA', 'var', 'cv', and 'mad'. See details.
`ignoreUnassignedVar`	logical indicating whether dimensionality reduction via top feature variability (i.e. 'var','cv','mad') should ignore unassigned samples in the primary clustering for calculation of the top features.

Details

The data matrix defined by `assay(x)` is transformed based on the transformation function defined in x. If `dimReduce="none"` the transformed matrix is returned. Otherwise, the user can request dimensionality reduction of the transformed data via `dimReduce`. 'PCA' refers to PCA of the transformed data with the top nPCADims kept. 'var', 'cv', and 'mad' refers to keeping the top most variable features, as defined by taking the variance, the mad, or the coefficient of variation (respectively) across all samples. nVarDims defines how many such features to keep for any of 'var','cv', or 'mad'; note that the number of features must be the same for all of these options (they cannot be set separately).

The PCA uses prcomp 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.

`ignoreUnassignedVar` has no impact for PCA reduction, which will always use all samples. At all times, regardless of the value of `ignoreUnassignedVar`, a matrix with the same number of columns of `assay(x)` (i.e. the same number of samples) will be returned.

`dimReduce`, `nPCADims`, `nVarDims` can all be a vector of values, in which case a list will be returned with the appropriate datasets as elements of the list.

Value

If `dimReduce`, `nPCADims`, `nVarDims` are all of length 1, a matrix will be returned of the same dimensions as `assay(x)`. If these arguments are vectors, then a list of data matrices will be return, each corresponding to the multiple choices implied by these parameters.

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

#transform and take top 3 dimensions
x <- transform(cc, dimReduce="PCA", nPCADims=3)

#transform and take return untransformed, top 5 features, and top 10 features
y <- transform(cc, dimReduce="var", nVarDims=c(NA, 5, 10))
names(y)

z<-transform(cc) #just return tranformed data
```

---

workflowClusters *Methods for workflow clusters*

Description

The main workflow of the package is made of `clusterMany`, `combineMany`, 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	which cluster to set to current in the workflow
eraseOld	logical. Only relevant if input x is of class ClusterExperiment. If TRUE, will erase existing workflow results (clusterMany as well as mergeClusters and combineMany). 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', 'combineMany') and iteration.

workflowClusterTable returns a table of how many of the clusterings belong to each of the following possible values: 'final', 'mergeClusters', 'combineMany' 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,nPCADims=c(5,10,50),  dimReduce="PCA",
clusterFunction="pam", ks=2:4, findBestK=c(FALSE), removeSil=TRUE,
subsample=FALSE)

clCommon <- combineMany(cl, whichClusters="workflow", proportion=0.7,
minSize=10)

clCommon <- makeDendrogram(clCommon)

clMerged <- mergeClusters(clCommon,mergeMethod="adjP")

head(workflowClusters(clMerged))
workflowClusterDetails(clMerged)
workflowClusterTable(clMerged)
```

Index