Package 'clusterExperiment'

October 17, 2017

Title Compare Clusterings for Single-Cell Sequencing

Version 1.2.0

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 (>= 3.3), methods, SummarizedExperiment

Imports NMF, RColorBrewer, ape, phylobase, cluster, stats, limma, dendextend, howmany, locfdr, matrixStats, graphics, parallel, MAST

Suggests BiocStyle, knitr, diagram, testthat, scRNAseq

VignetteBuilder knitr

LazyData true

RoxygenNote 6.0.1

biocViews Clustering, RNASeq, Sequencing, Software, SingleCell

NeedsCompilation no

R topics documented:

addClusters,ClusterExperiment,matrix-method 2
clusterContrasts,ClusterExperiment-method 3
clusterD 5
ClusterExperiment-class 8
ClusterExperiment-methods 11
clusterMany,matrix-method 13
clusterSingle 17
combineMany,matrix,missing-method 18
convertClusterLegend 20
getBestFeatures,matrix-method 21
makeDendrogram,ClusterExperiment-method 24
mergeClusters,matrix-method 26
nFeatures 28
nSamples 28

plotClusters,ClusterExperiment,character-method 29
plotHeatmap,SummarizedExperiment-method 33
plottingFunctions 37
RSEC 39
seqCluster 41
simData 44
subsampleClustering 45
transform 46
workflowClusters 48

Index **50**

`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, clusterFunction="pam", subsample=FALSE,
sequential=FALSE, clusterDArgs=list(k=3))

cl2 <- clusterSingle(simData, clusterFunction="pam", subsample=FALSE,
sequential=FALSE, clusterDArgs=list(k=5))

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.
`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 only option is the format appropriate for `limma` package, but we anticipate adding more.
`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

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")
```

`clusterD` *Cluster distance matrix from subsampling*

Description

Given a `n x n` matrix of distances, these functions will try to find the clusters based on the given clustering function. cluster01 and clusterK are internal functions and clusterD is a wrapper around these two functions for easier user interface. cluster01 and clusterK are not expected to be called directly by the user, except for ease in debugging user-defined clustering functions.

Usage

```
clusterD(x = NULL, diss = NULL, clusterFunction = c("hierarchical01",
  "tight", "pam", "hierarchicalK"), typeAlg = c("01", "K"),
  distFunction = NA, minSize = 1, orderBy = c("size", "best"),
  format = c("vector", "list"), clusterArgs = NULL, checkArgs = TRUE,
  returnD = FALSE, ...)

cluster01(diss, clusterFunction = c("hierarchical01", "tight"), alpha = 0.1,
  clusterArgs = NULL, checkArgs)

clusterK(diss, clusterFunction = c("pam", "hierarchicalK"),
  findBestK = FALSE, k, kRange, removeSil = FALSE, silCutoff = 0,
  clusterArgs = NULL, checkArgs)
```

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
`clusterFunction`	clusterFunction a function that clusters a nxn matrix of dissimilarities/distances. Can also be given character values to indicate use of internal wrapper functions for default methods. See Details for the format of what the function must take as arguments and what format the function must return.
`typeAlg`	character value of either '01' or 'K' determining whether the function given in clusterFunction should be called by clusterK or cluster01. Only used if clusterFunction is a user-defined function. Otherwise, for methods provided by the package (i.e. by user setting clusterFunction to a character value) clusterD will determine the appropriate input for 'typeAlg' and will ignore user input.

distFunction	a distance function to be applied to D. Only relevant if input D is a matrix of data, rather than a distance. See details.
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).
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.
clusterArgs	arguments to be passed directly to the clusterFunction, beyond the required input.
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.
returnD	logical as to whether to return the D matrix in output.
...	arguments given to clusterD to be passed to cluster01 or clusterK (depending on the value of typeAlg). Examples include 'k' for clusterK or 'alpha' for cluster01. These should not be the arguments needed by clusterFunction (which should be passed via the argument 'clusterArgs') but the actual arguments of cluster01 or clusterK.
alpha	a cutoff value of how much similarity needed for drawing blocks (lower values more strict).
findBestK	logical, whether should find best K based on average silhouette width (only used if clusterFunction of type "K").
k	single value to be used to determine how many clusters to find, if findBestK=FALSE (only used if clusterFunction of type "K").
kRange	vector of integers. If findBestK=TRUE, this gives the range of k's to look over. Default is k-2 to k+20, subject to those values being greater than 2. 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 when silhouette < silCutoff (only used if clusterFunction of type "K")
silCutoff	Requirement on minimum silhouette width to be included in cluster (only if removeSil=TRUE).

Details

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.

Types of algorithms: cluster01 is for clustering functions that expect as an input D that takes on 0-1 values (e.g. from subclustering). clusterK is for clustering functions that require an input k, the number of clusters, but arbitrary distance/dissimilarity matrix. cluster01 and clusterK are given as separate functions in order to allow the user to provide different clustering functions that

expect different types of input and for us to provide different shared processing of the results that is different for these different types of clustering methods (for example, removing low silhouette values is appropriate for clusterK clustering functions rather than cluster01 functions). It is also generally expected that cluster01 algorithms use the 0-1 nature of the input to set criteria as to where to find clusters and therefore do not need a pre-determined 'k'. On the other hand, clusterK functions are assumed to need a predetermined 'k' and are also assumed to cluster all samples to a cluster, and therefore clusterK gives options to exclude poorly clustered samples via silhouette distances.

cluster01 required format for input and output for clusterFunction: clusterFunction should be a function that takes (as a minimum) an argument "D" and "alpha". 0-1 clustering algorithms are expected to use the fact that the D input is 0-1 range to find the clusters, rather than a user defined number of clusters; "alpha" is the parameter that tunes the finding of such clusters. For example, a candidate block of samples might be considered a cluster if all values of D are greater than or equal to 1-alpha. The output is a list with each element corresponding to a cluster and the elements of the list corresponding to the indices of the samples that are in the cluster. The list is expected to be in order of 'best clusters' (as defined by the clusterFunction), with first being the best and last being worst.

cluster01 methods: "tight" method refers to the method of finding clusters from a subsampling matrix given internally in the tight algorithm code of Tsang and Wong. Arguments for the tight method are 'minSize.core' (default=2), which sets the minimimum number of samples that form a core cluster. "hierarchical01" refers to running the hclust algorithm on D and 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 'hierarchical' 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. Arguments of hclust can also be passed via clusterArgs to control the hierarchical clustering of D.

clusterK required format for input and output for clusterFunction: clusterFunction should be a function that takes as a minimum an argument 'D' and 'k'. The output must be a clustering, specified by integer values. The function `silhouette` will be used on the clustering to calculate silhouette scores for each observation.

clusterK methods: "pam" performs pam clustering on the input `D` matrix using `pam` in the cluster package. Arguments to `pam` can be passed via 'clusterArgs', except for the arguments 'x' and 'k' which are given by D and k directly. "hierarchicalK" performs hierarchical clustering on the input via the `hclust` and then applies `cutree` with the specified k to obtain clusters. Arguments to `hclust` can be passed via `clusterArgs`.

Value

clusterD returns a vector of cluster assignments (if format="vector") or a list of indices for each cluster (if format="list"). Clusters less than minSize are removed. If orderBy="size" the clusters are reordered by the size of the cluster, instead of by the internal ordering of the clusterFunction.

cluster01 and clusterK return a list of indices of the clusters found, which each element of the list corresponding to a cluster and the elements of that list a vector of indices giving the indices of the samples assigned to that cluster. Indices not included in any list are assumed to have not been clustered. The list is assumed to be ordered in terms of the 'best' cluster (as defined by the clusterFunction for cluster01 or by average silhoute for clusterK), for example in terms of most internal similarity of the elements, or average silhouette width.

Examples

```
data(simData)
cl1<-clusterD(simData,clusterFunction="pam",k=3)
cl2<-clusterD(simData,clusterFunction="hierarchical01")
cl3<-clusterD(simData,clusterFunction="tight")
#change distance to manhattan distance
cl4<-clusterD(simData,clusterFunction="pam",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 <- clusterD(simData, clusterFunction="hierarchical01", alpha=0.1,
minSize=5, clusterArgs=list(evalClusterMethod="average", method="single"))

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

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

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, coClustering = NULL)
```

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).
coClustering	matrix. Sets the 'coClustering' slot (see Slots).

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.

Value

A `ClusterExperiment` object.

Slots

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

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

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

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

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

`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.

`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 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'
clusterMatrix(x)

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

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

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 functions.

Details

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

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.

`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).

`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.

`clusterMany,matrix-method`

Create a matrix of clustering across values of parameters

Description

Given a range of parameters, this funciton 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, clusterDArgs = 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, ...)
```

Arguments

x	the data 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 meaning for different choices).
clusterFunction	function used for the clustering. Note that unlike in clusterSingle, this must be a character vector of pre-defined clustering techniques provided by clusterSingle, and can not be a user-defined function. Current functions are "tight", "hierarchical01","hierarchicalK", and "pam"
alphas	values of alpha to be tried. Only used for clusterFunctions of type '01' (either 'tight' or 'hierarchical01'). Determines tightness required in creating clusters from the dissimilarity matrix. Takes on values in [0,1]. See clusterD.
findBestK	logical, whether should find best K based on average silhouette width (only used if clusterFunction of type "K").
sequential	logical whether to use the sequential strategy (see details of seqCluster).
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 to get the distance matrix at each iteration; otherwise the distance function will be determined by argument distFunction passed in clusterDArgs (if input a data matrix).
silCutoff	Requirement on minimum silhouette width to be included in cluster (only if 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 clusterD 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 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 seqCluster.

minSizes	the minimimum size required for a cluster (in clusterD). Clusters smaller than this are not kept and samples are left unassigned.
verbose	logical. If TRUE it will print informative messages.
clusterDArgs	list of additional arguments to be passed to clusterD.
subsampleArgs	list of arguments to be passed to subsampleClustering.
seqArgs	list of additional 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

While the function allows for multiple values of clusterFunction, the code does not reuse the same subsampling matrix and try different clusterFunctions on it. If sequential=TRUE, different subsampleclusterFunctions 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 set in both subsampleArgs[["k"]] and clusterDArgs[["k"]] that are passed to clusterD and subsampleClustering. This passing of these arguments via subsampleArgs[["k"]] will only have an effect if 'subsample=TRUE'. Similarly, the passing of clusterDArgs[["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 clusterD unless kRange is specified by the user via the clusterDArgs; note this means that the default option of setting kRange that depends on the input k (see clusterD) is not available in clusterMany.

If the input is a ClusterExperiment object, currently existing orderSamples,coClustering or 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
- clusterDArgs a list of (possibly modified) arguments to clusterDArgs
- 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",
clusterDArgs=list(minSize=5, kRange=2:15), ncores=1, random.seed=48120))

## End(Not run)
```

clusterSingle *General wrapper method to cluster the data*

Description

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

Usage

```
## S4 method for signature 'matrixOrMissing,matrixOrMissing'
clusterSingle(x, diss,
  subsample = TRUE, sequential = FALSE, clusterFunction = c("tight",
  "hierarchical01", "pam", "hierarchicalK"), clusterDArgs = NULL,
  subsampleArgs = NULL, seqArgs = NULL, isCount = FALSE,
  transFun = NULL, dimReduce = c("none", "PCA", "var", "cv", "mad"),
  ndims = NA, clusterLabel = "clusterSingle")

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

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

Arguments

x	the data on which to run the clustering (features in rows).
diss	n x n data matrix of dissimilarities between the samples on which to run the clustering (only if subsample=FALSE)
subsample	logical as to whether to subsample via subsampleClustering to get the distance matrix at each iteration; otherwise the distance function will be determined by argument distFunction passed in clusterDArgs (if input a data matrix).
sequential	logical whether to use the sequential strategy (see details of seqCluster).
clusterFunction	passed to clusterD option 'clusterFunction' to indicate method of clustering, see clusterD.
clusterDArgs	list of additional arguments to be passed to clusterD.
subsampleArgs	list of arguments to be passed to subsampleClustering.
seqArgs	list of additional 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.
...	arguments to be passed on to the method for signature matrix.

Details

If sequential=TRUE, the sequential clustering controls the 'k' argument of the underlying clustering so setting 'k=' in the list given to clusterDArgs or subsampleArgs will not do anything and will produce a warning to that effect.

Value

A ClusterExperiment object.

See Also

clusterMany to compare multiple choices of parameters.

Examples

```
data(simData)

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

## End(Not run)

#use clusterSingle to do just clustering k=3 with no subsampling
clustNothing <- clusterSingle(simData, clusterFunction="pam",
subsample=FALSE, sequential=FALSE, clusterDArgs=list(k=3))
```

---

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 = 1,
  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 clusterD via alpha = 1 - proportion)
clusterFunction	the clustering to use (passed to clusterD); 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 clusterD
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 clusterD 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 clusterD on the proportion matrix with clusterFunction as the 01 clustering algorithm, alpha=1-proportion, minSize=minSize, and evalClusterMethod=c("average"). See help of clusterD 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 clusterD 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)
```

---

convertClusterLegend *Convert clusterLegend into useful formats*

Description

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

Usage

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

Arguments

object	a ClusterExperiment object.
output	character value, indicating desired type of conversion.

Details

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

Value

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.

---

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.
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.

Examples

```
data(simData)

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

`makeDendrogram,ClusterExperiment-method`

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("none", "PCA", "var", "cv",
  "mad"), ndims = NA, ignoreUnassignedVar = FALSE,
  unassignedSamples = c("outgroup", "cluster"), ...)

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

## S4 method for signature 'ClusterExperiment'
plotDendrogram(x, leaves = c("clusters",
  "samples"), clusterNames = TRUE, main, sub, ...)
```

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.

`leaves`
: if "samples" the dendrogram has one leaf per sample, otherwise it has one per cluster.

`clusterNames`
: logical. If `leaves="clusters"`, then clusters will be identified with their 'name' value in legend; otherwise the 'clusterIds' value will be used.

`main`
: passed to the `plot` function.

`sub`
: passed to the `plot` function.

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 `leaves="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

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, clusterFunction="pam", subsample=FALSE,
sequential=FALSE, clusterDArgs=list(k=8))
```

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

`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", "adjP", "locfdr", "MB", "JC"),
  plotType = c("none", "all", "mergeMethod", "adjP", "locfdr", "MB", "JC"),
  cutoff = 0.1, doPlot = TRUE, isCount = TRUE, ...)

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

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; The default for matrix (NULL) is to recalculate it with the given (x, cl) pair. If x is a `ClusterExperiment` object, the dendrogram in the slot `dendro_clusters` will be used. 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.
`plotType`	what type of plotting of dendrogram. If 'all', then all the estimates of proportion non-null will be plotted; if 'mergeMethod', then only the value used in the merging is plotted for each node.
`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.
`doPlot`	logical as to whether to plot the dendrogram (overrides `plotType` value). Mainly used for internal coding purposes.
`isCount`	logical as to whether input data is a count matrix. See details.

...	for signature matrix, arguments passed to the plot.phylo function of ade4 that plots the dendrogram. For signature ClusterExperiment arguments passed to the method for signature matrix.
eraseOld	logical. Only relevant if input x is of class ClusterExperiment. If TRUE, will erase existing workflow results (clusterMany as well as mergeClusters and 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.

Details

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.

"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/Nulland "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.

If mergeMethod is not equal to 'none' then the plotting will indicate where the clusters will be merged (assuming plotType is not 'none').

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).

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.

Examples

```
data(simData)

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

#make dendrogram
cl <- makeDendrogram(cl)
```

```
#merge clusters with plotting. Note argument 'use.edge.length' can improve
#readability
merged <- mergeClusters(cl, plot=TRUE, plotType="all",
mergeMethod="adjP", use.edge.length=FALSE)

#compare merged to original
table(primaryCluster(cl), primaryCluster(merged))
```

---

nFeatures *Generic function that returns the number of features*

Description

Given an object that describes a dataset or a model, it returns the number of features.

Usage

```
nFeatures(x)
```

Arguments

x	an object that describes a dataset or a model.

Value

the number of features.

---

nSamples *Generic function that returns the number of samples*

Description

Given an object that describes a model or a dataset, it returns the number of samples.

Usage

```
nSamples(x)
```

Arguments

x	an object that describes a dataset or a model.

Value

the number of samples.

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(clusters,
  whichClusters = c("workflow", "all"), ...)

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

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

## S4 method for signature 'matrix,missing'
plotClusters(clusters, 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"),
  clNames = colnames(clusters), add = FALSE, xCoord = NULL, ylim = NULL,
  tick = FALSE, ylab = "", xlab = "", axisLine = 0, box = FALSE, ...)
```

Arguments

clusters	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 clusterTypess to be used; 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.

`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).

`clNames` names to go with the columns (clusterings) in matrix `colorMat`.

`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).

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)
```

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("hclust", "dendrogramValue", "orderSamplesValue",
  "primaryCluster"), clusterFeaturesData = c("var", "all", "PCA"),
  nFeatures = NULL, visualizeData = c("transformed", "centeredAndScaled",
  "original"), whichClusters = c("primary", "workflow", "all", "none"),
  sampleData = NULL, clusterFeatures = TRUE, colorScale, ...)

## 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, ...)

## 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.
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, either a matrix that will be used to in 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, the input should be either character or integers or logical. 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 dimensions of assay(data).

whichClusters
: character string, or vector of characters or integers, indicating what clusters should be visualized with the heatmap.

sampleData
: If input 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 is matrix, sampleData is a matrix of additional data on the samples to show above heatmap. 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).

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.

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.

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.

`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 missing, a new one will be created based on the `primaryCluster` of data. If equal to "hclust", then standard hierachical clustering of the transformed data will be used. If 'orderSamplesValue' no clustering of the samples will be done, and instead the samples will be ordered as in the slot `orderSamples` of `data`. If 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`. If not one of these values, `clusterSamplesData` can be a character vector matching the clusterLabels (colnames of clusterMatrix).

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 consider 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...).

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

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` *Various functions useful for plotting*

Description

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)

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.)
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.

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 unique. However, they can be given to the `labRow` argument of `aheatmap` or `plotHeatmap`.

Examples

```
data(simData)

x <- makeBlankData(simData[,1:10], groupsOfFeatures=list(c(5, 2, 3), c(20,
34, 25)))
showBigPalette()
setBreaks(data=simData,breaks=.9)

#show the palette colors
showHeatmapPalettes()

#compare the palettes on heatmap
cl <- clusterSingle(simData, clusterFunction="pam", subsample=FALSE,
sequential=FALSE, clusterDArgs=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 = c("tight", "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,
  clusterDArgs = NULL, subsampleArgs = NULL, seqArgs = NULL, ncores = 1,
  random.seed = NULL, run = TRUE)

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

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

Arguments

x	the data 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 provided by clusterSingle, and can not be a user-defined function. Current functions are "tight", "hierarchical01","hierarchicalK", and "pam"
alphas	values of alpha to be tried. Only used for clusterFunctions of type '01' (either 'tight' or 'hierarchical01'). Determines tightness required in creating clusters from the dissimilarity matrix. Takes on values in [0,1]. See clusterD.
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 seqCluster.
minSizes	the minimimum size required for a cluster (in clusterD). 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.
clusterDArgs	list of additional arguments to be passed to clusterD.
subsampleArgs	list of arguments to be passed to subsampleClustering.
seqArgs	list of additional 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.

---

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, clusterFunction = c("tight",
  "hierarchical01", "pam", "hierarchicalK"), subsample = TRUE, beta = 0.7,
  top.can = 15, remain.n = 30, k.min = 3, k.max = k0 + 10,
  verbose = TRUE, subsampleArgs = NULL, clusterDArgs = NULL)
```

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.
clusterFunction	passed to clusterDMat option 'clusterFunction' to indicate method of clustering, see clusterD.
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 clusterD.
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 clusterD (ranked by 'orderBy' argument given to clusterD) 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. 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.
clusterDArgs	list of arguments to be passed to clusterD(which can include arguments to be passed to cluster01 or clusterK).

Details

This code is adapted from the code of the tightClust package of Tseng and Wong

Each iteration of the algorithm will cluster the current set of samples. Depending on the method, the number of clusters resulting from clusterD may not be equal to the K used in the clustering of the (subsampled) data. The resulting clusters will then be compared to clusters found in the previous iteration that set the subsampling clustering to 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 a cluster in the current iteration that has overlap similarity > beta to a cluster in the previous iteration, then the cluster with the largest such similarity will be identified as a 'final' cluster and the samples in it will be removed for future iterations. The algorithm will then continue to the next iteration, but without these samples. Furthermore, in this case K for the next iteration will NOT be set to K+1, but will be reset to kinit-1, where kinit was the first K used after the previous 'final' cluster was removed. If kinit-1<k.min, then K will be set to k.min.

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

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 'final' cluster after the algorithm stops, will be classified as unclustered (given a value of -1)

'subsample' controls what is the D (distance) matrix used for clustering at each iteration. If subsample=TRUE, D is given via `subsampleClustering` function with k=K (with additional arguments passed via subsampleArgs). If subsample=FALSE, D is dist(x), for the samples currently considered in the iteration and clusterFunction must be of the 'K' type (e.g. "pam", see `clusterD`) or an error will be produced. The nsample x nsample matrix D is then clustered via `clusterD` to find clusters. The option 'clusterFunction' is passed to the argument 'clusterFunction' of `clusterD` to control what method is used to cluster D.

If clusterFunction is of type 'K' (e.g. "pam", see `clusterD`) the 'k' argument of `clusterK` called by `clusterD` is set to the current iteration of K by the sequential iteration, so setting 'k=' in the list given to clusterDArgs will not do anything and will produce a warning to that effect.

Similarly, the current K of the iteration also 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.

If subsample=FALSE and 'findBestK=FALSE' is passed to clusterDArgs, then each iteration will run the clustering given by clusterFunction on dist(x) iterating over k. However, if subsample=FALSE, you should not set 'findBestK=TRUE' (otherwise clustering dist(x) will be essentially the same for iterating over different k and there is no method implemented to change the choice of how to remove a cluster other than similarity as you change k); an error message will be given if this combination of options are set.

However, if clusterFunction="pam" (or is of type 'K') and subsample=TRUE passing either 'findBestK=TRUE' or 'findBestK=FALSE' will function as expected. In particular, the iteration over K will set the number of clusters for clustering of each subsample. If findBestK=FALSE, that same K will be used for clustering of DMat. If findBestK=TRUE, then `clusterD` will search for best k; note that the default 'kRange' over which `clusterD` searches when findBestK=TRUE depends on the input value of 'k' (you can change this to a fixed set of values by setting 'kRange' explicitly in the clusterDArgs list).

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

Examples

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

set.seed(12908)

clustSeqHier <- seqCluster(t(simData), k0=5, subsample=TRUE,
clusterFunction="hierarchical01", beta=0.8, subsampleArgs=list(resamp.n=100,
samp.p=0.7, clusterFunction="kmeans", clusterArgs=list(nstart=10)),
clusterDArgs=list(minSize=5))

## 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:
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")
```

`subsampleClustering` *Cluster subsamples of the data*

Description

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

Usage

```
subsampleClustering(x, k, clusterFunction = "pam", clusterArgs = NULL,
  classifyMethod = c("All", "InSample", "OutOfSample"),
  classifyFunction = NULL, resamp.num = 100, samp.p = 0.7, ncores = 1,
  ...)
```

Arguments

`x`	the data on which to run the clustering (samples in columns).
`k`	number of clusters to find for each clustering of a subsample (passed to clusterFunction).
`clusterFunction`	a function that clusters a `p x n` matrix of data. Can also be given character values 'pam' or 'kmeans' to indicate use of internal wrapper functions. Must accept arguments 'x' and 'k' (whether uses them or not). See Details for format of what must return.
`clusterArgs`	a list of parameter arguments to be passed to clusterFunction.
`classifyMethod`	method for determining which samples should be used in the co-occurance matrix. "All"= all samples, "OutOfSample"= those not subsampled, and "InSample"=those in the subsample. "All" and "OutOfSample" require that you provide classifyFunction to define how to classify those samples not in the subsample

into a cluster. If "All" is chosen, all samples will be classified into clusters via the classifyFunctions, not just those that are out-of-sample. Note if not choose 'All' possible to get NAs in resulting D matrix (particularly if not enough subsamples taken).

`classifyFunction` a function which, given the output of clusterFunction and new data points, will classify the new data points into a cluster.

`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.

`...` arguments passed to mclapply (if ncores>1).

Details

The `clusterFunction` must be a function that takes as an argument 'x' which is a `p x n` matrix of data and integer 'k'. It minimally must return a list with element named 'clustering' giving the vector of cluster ids. To be incorporated with the larger hierarchy, it should be list with elements of a partition object, just as is returned by `pam`. Generally, the user will need to write a wrapper function to do this. In the case of pam or kmeans, the user can identify clusterFunction as "pam" or "kmeans", and the package functions will use internally written wrappers for the clusterFunction and classifyFunction arguments. Additional arguments should be supplied via clusterArgs.

The classifyFunction should take as an object a data matrix 'x' with samples on the columns, and the output of the clusterFunction. 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).

Value

A `n x n` matrix of co-occurances.

Examples

```
data(simData)

subD <- subsampleClustering(t(simData), k=3, clusterFunction="kmeans",
clusterArgs=list(nstart=10), resamp.n=100, samp.p=0.7)

heatmap(subD)
```

---

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(x, nPCADims = NA, nVarDims = NA,
  dimReduce = "none", ignoreUnassignedVar = FALSE)
```

Arguments

x	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

∗Topic **datasets**
plottingFunctions, 37
∗Topic **data**
simData, 44
[,ClusterExperiment,ANY,ANY,ANY-method *(*ClusterExperiment-methods*)*, 11
[,ClusterExperiment,ANY,character,ANY-method *(*ClusterExperiment-methods*)*, 11
[,ClusterExperiment,ANY,logical,ANY-method *(*ClusterExperiment-methods*)*, 11
[,ClusterExperiment,ANY,numeric,ANY-method *(*ClusterExperiment-methods*)*, 11

addClusters *(*addClusters,ClusterExperiment,matrix-method*)*, 2
addClusters,ClusterExperiment,ClusterExperiment-method *(*addClusters,ClusterExperiment,matrix-method*)*, 2
addClusters,ClusterExperiment,matrix-method, 2
addClusters,ClusterExperiment,numeric-method *(*addClusters,ClusterExperiment,matrix-method*)*, 2
aheatmap, *21*, *33–36*, *39*

bigPalette *(*plottingFunctions*)*, 37

cluster01, *42*
cluster01 *(*clusterD*)*, 5
clusterContrasts, *22*
clusterContrasts *(*clusterContrasts,ClusterExperiment-method*)*, 3
clusterContrasts,ClusterExperiment-method, 3
clusterContrasts,vector-method *(*clusterContrasts,ClusterExperiment-method*)*, 3
clusterD, 5, *14, 15*, *17*, *19*, *40–43*
ClusterExperiment, *2, 3*, *12*, *17, 18*, *20*, *22*, *24*, *26, 27*, *29*, *31*, *33, 34*, *48*
ClusterExperiment *(*ClusterExperiment-class*)*, 8
clusterExperiment, *19*
clusterExperiment *(*ClusterExperiment-class*)*, 8
clusterExperiment,matrix,ANY-method *(*ClusterExperiment-class*)*, 8
clusterExperiment,SummarizedExperiment,character-method *(*ClusterExperiment-class*)*, 8
clusterExperiment,SummarizedExperiment,factor-method *(*ClusterExperiment-class*)*, 8
clusterExperiment,SummarizedExperiment,matrix-method *(*ClusterExperiment-class*)*, 8
clusterExperiment,SummarizedExperiment,numeric-method *(*ClusterExperiment-class*)*, 8
ClusterExperiment-class, 8
clusterExperiment-methods, 11
clusterInfo *(*ClusterExperiment-methods*)*, 11
clusterInfo,ClusterExperiment-method *(*ClusterExperiment-methods*)*, 11
clusterK, *42, 43*
clusterK *(*clusterD*)*, 5
clusterLabels *(*ClusterExperiment-methods*)*, 11
clusterLabels,ClusterExperiment-method *(*ClusterExperiment-methods*)*, 11
clusterLabels<- *(*ClusterExperiment-methods*)*, 11
clusterLabels<-,ClusterExperiment,character-method *(*ClusterExperiment-methods*)*, 11
clusterLegend *(*ClusterExperiment-methods*)*, 11
clusterLegend,ClusterExperiment-method *(*ClusterExperiment-methods*)*, 11
clusterLegend<- *(*ClusterExperiment-methods*)*, 11
clusterLegend<-,ClusterExperiment,list-method *(*ClusterExperiment-methods*)*, 11
clusterMany, *8*, *18*, *48*
clusterMany *(*clusterMany,matrix-method*)*, 13
clusterMany,ClusterExperiment-method *(*clusterMany,matrix-method*)*, 13
clusterMany,list-method

(clusterMany,matrix-method), 13
clusterMany,matrix-method, 13
clusterMany,SummarizedExperiment-method (clusterMany,matrix-method), 13
clusterMatrix (ClusterExperiment-methods), 11
clusterMatrix,ClusterExperiment-method (ClusterExperiment-methods), 11
clusterMatrixNamed (ClusterExperiment-methods), 11
clusterMatrixNamed,ClusterExperiment-method (ClusterExperiment-methods), 11
clusterSingle, *8*, *10*, *14*, *16*, 17, *35*, *40*
clusterSingle,ClusterExperiment,missing-method (clusterSingle), 17
clusterSingle,ClusterExperiment-method (clusterSingle), 17
clusterSingle,matrix,missing-method (clusterSingle), 17
clusterSingle,matrix-method (clusterSingle), 17
clusterSingle,matrixOrMissing,matrixOrMissing-method (clusterSingle), 17
clusterSingle,SummarizedExperiment,missing-method (clusterSingle), 17
clusterSingle-methods (clusterSingle), 17
clusterTypes (ClusterExperiment-methods), 11
clusterTypes,ClusterExperiment-method (ClusterExperiment-methods), 11
clusterTypes<- (ClusterExperiment-methods), 11
clusterTypes<-,ClusterExperiment,character-method (ClusterExperiment-methods), 11
coClustering (ClusterExperiment-methods), 11
coClustering,ClusterExperiment-method (ClusterExperiment-methods), 11
coClustering<- (ClusterExperiment-methods), 11
coClustering<-,ClusterExperiment,matrix-method (ClusterExperiment-methods), 11
combineMany, *8*, *40*, *48*
combineMany (combineMany,matrix,missing-method), 18
combineMany,ClusterExperiment,character-method (combineMany,matrix,missing-method), 18
combineMany,ClusterExperiment,missing-method (combineMany,matrix,missing-method), 18
combineMany,ClusterExperiment,numeric-method (combineMany,matrix,missing-method), 18
combineMany,matrix,missing-method, 18
ConsensusClusterPlus, *31*
convertClusterLegend, 20
convertClusterLegend,ClusterExperiment-method (convertClusterLegend), 20
cutree, *7*

getBestFeatures, *27*
getBestFeatures (getBestFeatures,matrix-method), 21
getBestFeatures,ClusterExperiment-method (getBestFeatures,matrix-method), 21
getBestFeatures,matrix-method, 21

hclust, *7*, *25*
howmany, *27*

limma, *4*, *22*
locfdr, *27*

makeBlankData, *36*
makeBlankData (plottingFunctions), 37
makeContrasts, *4*
makeDendrogram, *10*, *26*, *27*, *40*
makeDendrogram (makeDendrogram,ClusterExperiment-method), 24
makeDendrogram,ClusterExperiment-method, 24
makeDendrogram,matrix-method (makeDendrogram,ClusterExperiment-method), 24
mergeClusters, *40*, *41*, *48*
mergeClusters (mergeClusters,matrix-method), 26
mergeClusters,ClusterExperiment-method (mergeClusters,matrix-method), 26
mergeClusters,matrix-method, 26

nClusters (ClusterExperiment-methods), 11
nClusters,ClusterExperiment-method (ClusterExperiment-methods), 11
nFeatures, 28
nFeatures,ClusterExperiment-method (ClusterExperiment-methods), 11

nSamples, 28
nSamples,ClusterExperiment-method (ClusterExperiment-methods), 11

orderSamples (ClusterExperiment-methods), 11
orderSamples,ClusterExperiment-method (ClusterExperiment-methods), 11
orderSamples<- (ClusterExperiment-methods), 11
orderSamples<-,ClusterExperiment,numeric-method (ClusterExperiment-methods), 11

pam, *7*, *46*
plot, *29*
plot.dendrogram, *25*
plot.phylo, *27*
plotClusters, *38*
plotClusters (plotClusters,ClusterExperiment,character-method), 29
plotClusters,ClusterExperiment,character-method, 29
plotClusters,ClusterExperiment,missing-method (plotClusters,ClusterExperiment,character-method), 29
plotClusters,ClusterExperiment,numeric-method (plotClusters,ClusterExperiment,character-method), 29
plotClusters,matrix,missing-method (plotClusters,ClusterExperiment,character-method), 29
plotCoClustering (plotHeatmap,SummarizedExperiment-method), 33
plotCoClustering,ClusterExperiment-method (plotHeatmap,SummarizedExperiment-method), 33
plotDendrogram (makeDendrogram,ClusterExperiment-method), 24
plotDendrogram,ClusterExperiment-method (makeDendrogram,ClusterExperiment-method), 24
plotHeatmap, *39*
plotHeatmap (plotHeatmap,SummarizedExperiment-method), 33
plotHeatmap,ClusterExperiment-method (plotHeatmap,SummarizedExperiment-method), 33
plotHeatmap,matrix-method (plotHeatmap,SummarizedExperiment-method), 33
plotHeatmap,SummarizedExperiment-method, 33
plottingFunctions, 37
primaryCluster (ClusterExperiment-methods), 11
primaryCluster,ClusterExperiment-method (ClusterExperiment-methods), 11
primaryClusterIndex (ClusterExperiment-methods), 11
primaryClusterIndex,ClusterExperiment-method (ClusterExperiment-methods), 11
primaryClusterIndex<- (ClusterExperiment-methods), 11
primaryClusterIndex<-,ClusterExperiment,numeric-method (ClusterExperiment-methods), 11
primaryClusterNamed (ClusterExperiment-methods), 11
primaryClusterNamed,ClusterExperiment-method (ClusterExperiment-methods), 11

removeClusters (addClusters,ClusterExperiment,matrix-method), 2
removeClusters,ClusterExperiment,character-method (addClusters,ClusterExperiment,matrix-method), 2
removeClusters,ClusterExperiment,numeric-method (addClusters,ClusterExperiment,matrix-method), 2
removeUnclustered (addClusters,ClusterExperiment,matrix-method), 2
removeUnclustered,ClusterExperiment-method (addClusters,ClusterExperiment,matrix-method), 2
RSEC, 39
RSEC,ClusterExperiment-method (RSEC), 39
RSEC,matrix-method (RSEC), 39
RSEC,SummarizedExperiment-method (RSEC), 39
RSEC-methods (RSEC), 39

seqCluster, *14*, *15*, *17*, *40*, *41*, 41
seqPal1 (plottingFunctions), 37
seqPal2 (plottingFunctions), 37
seqPal3 (plottingFunctions), 37
seqPal4 (plottingFunctions), 37
seqPal5 (plottingFunctions), 37
setBreaks, *35*
setBreaks (plottingFunctions), 37
setToCurrent (workflowClusters), 48

setToCurrent,ClusterExperiment-method
(workflowClusters), 48
setToFinal (workflowClusters), 48
setToFinal,ClusterExperiment-method
(workflowClusters), 48
show,ClusterExperiment-method
(ClusterExperiment-methods), 11
showBigPalette (plottingFunctions), 37
showHeatmapPalettes
(plottingFunctions), 37
silhouette, *7*
simCount (simData), 44
simData, 44
subsampleClustering, *14, 15*, *17*, *41–43*, 45
SummarizedExperiment, *3*, *12*, *17*, *33*

topTable, *22, 23*
topTableF, *22*
transform, *14*, *17*, *24*, *40*, 46
transform,ClusterExperiment-method
(transform), 46
transformation
(ClusterExperiment-methods), 11
transformation,ClusterExperiment-method
(ClusterExperiment-methods), 11
trueCluster (simData), 44

voom, *22*

workflowClusterDetails
(workflowClusters), 48
workflowClusterDetails,ClusterExperiment-method
(workflowClusters), 48
workflowClusters, *29*, 48
workflowClusters,ClusterExperiment-method
(workflowClusters), 48
workflowClusterTable
(workflowClusters), 48
workflowClusterTable,ClusterExperiment-method
(workflowClusters), 48