Package 'celda'

October 16, 2019

Title CEllular Latent Dirichlet Allocation

Version 1.0.4

Description celda leverages Bayesian hierarchical modeling to cluster genes, cells, or both simultaneously from single cell sequencing data.

Depends R (>= 3.6)

VignetteBuilder knitr

Imports stats, plyr, foreach, ggplot2, RColorBrewer, grid, scales, gtable, grDevices, graphics, matrixStats, doParallel, digest, gridExtra, methods, reshape2, MAST, S4Vectors, data.table, Rcpp, RcppEigen, umap, enrichR, stringi, SummarizedExperiment, MCMCprecision, ggrepel, Rtsne, withr

Suggests testthat, knitr, roxygen2, rmarkdown, corrplot, Matrix, biomaRt, covr, M3DExampleData, BiocManager, BiocStyle

LinkingTo Rcpp, RcppEigen

License MIT + file LICENSE

Encoding UTF-8

LazyData false

RoxygenNote 6.1.1

BugReports https://github.com/campbio/celda/issues

biocViews SingleCell, GeneExpression, Clustering, Sequencing, Bayesian

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

git_branch RELEASE_3_9

git_last_commit e2cf9f9

git_last_commit_date 2019-06-03

Date/Publication 2019-10-15

Author Joshua Campbell [aut, cre],
Sean Corbett [aut],
Yusuke Koga [aut],
Zhe Wang [aut]

Maintainer Joshua Campbell <camp@bu.edu>

R topics documented:

`appendCeldaList` *Append two celdaList objects*

Description

Returns a single celdaList representing the combination of two provided celdaList objects.

Usage

```
appendCeldaList(list1, list2)
```

Arguments

`list1`	A celda_list object
`list2`	A celda_list object to be joined with list_1

Value

A celdaList object. This object contains all resList entries and runParam records from both lists.

Examples

```
data(celdaCGGridSearchRes)
appendedList <- appendCeldaList(celdaCGGridSearchRes,
    celdaCGGridSearchRes)
```

`availableModels` *available models*

Description

available models

Usage

```
availableModels
```

Format

An object of class `character` of length 3.

Examples

```
data(availableModels)
availableModels
```

bestLogLikelihood *Get the log-likelihood*

Description

Retrieves the final log-likelihood from all iterations of Gibbs sampling used to generate a celdaModel.

Usage

```
bestLogLikelihood(celdaMod)
```

Arguments

celdaMod	A celdaModel object of class celda_C, celda_G, or celda_CG.

Value

Numeric. The log-likelihood at the final step of Gibbs sampling used to generate the model.

Examples

```
data(celdaCGMod)
bestLogLikelihood(celdaCGMod)
```

bestLogLikelihood,celdaModel-method *Get the log-likelihood*

Description

Retrieves the final log-likelihood from all iterations of Gibbs sampling used to generate a celdaModel.

Usage

```
## S4 method for signature 'celdaModel'
bestLogLikelihood(celdaMod)
```

Arguments

celdaMod	A celdaModel object of class celda_C, celda_G, or celda_CG.

Value

Numeric. The log-likelihood at the final step of Gibbs sampling used to generate the model.

Examples

```
data(celdaCGMod)
bestLogLikelihood(celdaCGMod)
```

celda *Celda models*

Description

List of available Celda models with correpsonding descriptions.

Usage

```
celda()
```

Value

None

Examples

```
celda()
```

celdaCGGridSearchRes *celdaCGGridSearchRes*

Description

Example results of celdaGridSearch on celdaCGSim

Usage

```
celdaCGGridSearchRes
```

Format

An object as returned from celdaGridSearch()

Examples

```
data(celdaCGSim)
celdaCGGridSearchRes = celdaGridSearch(celdaCGSim$counts,
    model = "celda_CG",
    paramsTest = list(K = seq(4, 6), L = seq(9, 11)),
    paramsFixed = list(sampleLabel = celdaCGSim$sampleLabel),
    bestOnly = TRUE,
    nchains = 1,
    cores = 2)
```

celdaCGMod *celdaCGmod*

Description

celda_CG results generated from celdaCGSim

Usage

```
celdaCGMod
```

Format

A celda_CG object

Examples

```
data(celdaCGSim)
celdaCGMod = celda_CG(celdaCGSim$counts,
    K = celdaCGSim$K,
    L = celdaCGSim$L,
    nchains = 1)
```

celdaCGSim *celdaCGSim*

Description

An example simulated count matrix from the celda_CG model.

Usage

```
celdaCGSim
```

Format

A list of counts and properties as returned from simulateCells().

Examples

```
celdaCGSim = simulateCells("celda_CG")
```

celdaCMod *celdaCMod*

Description

celda_C results generated from celdaCSim

Usage

```
celdaCMod
```

Format

A celda_C object

Examples

```
data(celdaCSim)
celdaCMod = celda_C(celdaCSim$counts, K = celdaCSim$K, nchains = 1)
```

celdaCSim *celdaCSim*

Description

An example simulated count matrix from the celda_C model.

Usage

```
celdaCSim
```

Format

A list of counts and properties as returned from simulateCells().

Examples

```
celdaCSim = simulateCells("celda_C")
```

celdaGMod *celdaGMod*

Description

celda_G results generated from celdaGsim

Usage

```
celdaGMod
```

Format

A celda_G object

Examples

```
data(celdaGSim)
celdaGMod = celda_G(celdaGSim$counts, L = celdaGSim$L, nchains = 1)
```

celdaGridSearch *Run Celda in parallel with multiple parameters*

Description

Run Celda with different combinations of parameters and multiple chains in parallel. The variable 'availableModels' contains the potential models that can be utilized. Different parameters to be tested should be stored in a list and passed to the argument 'paramsTest'. Fixed parameters to be used in all models, such as 'sampleLabel', can be passed as a list to the argument 'paramsFixed'. When 'verbose = TRUE', output from each chain will be sent to a log file but not be displayed in stdout.

Usage

```
celdaGridSearch(counts, model, paramsTest, paramsFixed = NULL,
  maxIter = 200, nchains = 3, cores = 1, bestOnly = TRUE,
  perplexity = TRUE, verbose = TRUE, logfilePrefix = "Celda")
```

Arguments

counts	Integer matrix. Rows represent features and columns represent cells.
model	Celda model. Options available in 'celda::availableModels'.
paramsTest	List. A list denoting the combinations of parameters to run in a celda model. For example, 'list(K = seq(5, 10), L = seq(15, 20))' will run all combinations of K from 5 to 10 and L from 15 to 20 in model 'celda_CG()'.
paramsFixed	List. A list denoting additional parameters to use in each celda model. Default NULL.
maxIter	Integer. Maximum number of iterations of sampling to perform. Default 200.
nchains	Integer. Number of random cluster initializations. Default 3.

cores	Integer. The number of cores to use for parallel estimation of chains. Default 1.
bestOnly	Logical. Whether to return only the chain with the highest log likelihood per combination of parameters or return all chains. Default TRUE.
perplexity	Logical. Whether to calculate perplexity for each model. If FALSE, then perplexity can be calculated later with 'resamplePerplexity()'. Default TRUE.
verbose	Logical. Whether to print log messages during celda chain execution. Default TRUE.
logfilePrefix	Character. Prefix for log files from worker threads and main process. Default "Celda".

Value

Object of class 'celdaList', which contains results for all model parameter combinations and summaries of the run parameters

See Also

'celda_G()' for feature clustering, 'celda_C()' for clustering of cells, and 'celda_CG()' for simultaneous clustering of features and cells. 'subsetCeldaList()' can subset the 'celdaList' object. 'selectBestModel()' can get the best model for each combination of parameters.

Examples

```
data(celdaCGSim)
#' ## Run various combinations of parameters with 'celdaGridSearch'
celdaCGGridSearchRes <- celdaGridSearch(celdaCGSim$counts,
    model = "celda_CG",
    paramsTest = list(K = seq(4, 6), L = seq(9, 11)),
    paramsFixed = list(sampleLabel = celdaCGSim$sampleLabel),
    bestOnly = TRUE,
    nchains = 1,
    cores = 2)
```

`celdaGSim` *celdaGSim*

Description

An example simulated count matrix from the celda_G model.

Usage

```
celdaGSim
```

Format

A list of counts and properties as returned from simulateCells()

Examples

```
celdaGSim = simulateCells("celda_G")
```

`celdaHeatmap` *Plot celda Heatmap*

Description

Render a stylable heatmap of count data based on celda clustering results.

Usage

```
celdaHeatmap(counts, celdaMod, featureIx, ...)
```

Arguments

`counts`	Integer matrix. Rows represent features and columns represent cells. This matrix should be the same as the one used to generate 'celdaMod'.
`celdaMod`	A celdaModel object of class "celda_C", "celda_G", or "celda_CG".
`featureIx`	Integer vector. Select features for display in heatmap. If NULL, no subsetting will be performed. Default NULL.
`...`	Additional parameters.

Value

list A list containing dendrogram information and the heatmap grob

Examples

```
data(celdaCGSim, celdaCGMod)
celdaHeatmap(celdaCGSim$counts, celdaCGMod)
```

`celdaHeatmap,celda_C-method`
Heatmap for celda_C

Description

Renders an expression heatmap to visualize 'celda_C()' results. Features to include in the heatmap must be supplied.

Usage

```
## S4 method for signature 'celda_C'
celdaHeatmap(counts, celdaMod, featureIx, ...)
```

Arguments

`counts`	Integer matrix. Rows represent features and columns represent cells. This matrix should be the same as the one used to generate 'celdaMod'.
`celdaMod`	Celda object of class 'celda_C'.
`featureIx`	Integer vector. Indices of features to plot, such the top features from a differential expression analysis.
`...`	Additional parameters.

Value

list A list containing dendrograms and the heatmap grob

See Also

'celda_C()' for clustering cells and 'celdaTsne()' for generating 2-dimensional coordinates

Examples

```
data(celdaCSim, celdaCMod)
celdaHeatmap(celdaCSim$counts, celdaCMod)
```

`celdaHeatmap,celda_CG-method`
Heatmap for celda_CG

Description

Renders an expression heatmap to visualize 'celda_CG()' results. The top 'nfeatures' for each module will be included in the heatmap.

Usage

```
## S4 method for signature 'celda_CG'
celdaHeatmap(counts, celdaMod, nfeatures = 25, ...)
```

Arguments

`counts`	Integer matrix. Rows represent features and columns represent cells. This matrix should be the same as the one used to generate. 'celdaMod'.
`celdaMod`	Celda object of class 'celda_CG'.
`nfeatures`	Integer. Maximum number of features to select for each module. Default 25.
`...`	Additional parameters.

Value

A list containing dendrograms and the heatmap grob

See Also

'celda_CG()' for clustering features and cells and 'celdaTsne()' for generating 2-dimensional coordinates.

Examples

```
data(celdaCGSim, celdaCGMod)
celdaHeatmap(celdaCGSim$counts, celdaCGMod)
```

celdaHeatmap,celda_G-method *Heatmap for celda_CG*

Description

Renders an expression heatmap to visualize 'celda_CG()' results. The top 'nfeatures' for each module will be included in the heatmap.

Usage

```
## S4 method for signature 'celda_G'
celdaHeatmap(counts, celdaMod, nfeatures = 25, ...)
```

Arguments

counts	Integer matrix. Rows represent features and columns represent cells. This matrix should be the same as the one used to generate 'celdaMod'.
celdaMod	Celda object of class 'celda_G'.
nfeatures	Integer. Maximum number of features to select for each module. Default 25.
...	Additional parameters.

Value

list A list containing the dendrograms and the heatmap grob.

See Also

'celda_G()' for clustering features and 'celdaTsne()' for generating 2-dimensional coordinates.

Examples

```
data(celdaGSim, celdaGMod)
celdaHeatmap(celdaGSim$counts, celdaGMod)
```

celdaPerplexity *Get perplexity for every model in a celdaList*

Description

Returns perplexity for each model in a celdaList as calculated by 'perplexity().'

Usage

```
celdaPerplexity(celdaList)
```

Arguments

celdaList	An object of class celdaList.

Value

List. Contains one celdaModel object for each of the parameters specified in the 'runParams()' of the provided celda list.

Examples

```
data(celdaCGGridSearchRes)
celdaCGGridModelPerplexities <- celdaPerplexity(celdaCGGridSearchRes)
```

---

celdaPerplexity,celdaList-method

Get perplexity for every model in a celdaList

Description

Returns perplexity for each model in a celdaList as calculated by 'perplexity().'

Usage

```
## S4 method for signature 'celdaList'
celdaPerplexity(celdaList)
```

Arguments

celdaList	An object of class celdaList.

Value

List. Contains one celdaModel object for each of the parameters specified in the 'runParams()' of the provided celda list.

Examples

```
data(celdaCGGridSearchRes)
celdaCGGridModelPerplexities <- celdaPerplexity(celdaCGGridSearchRes)
```

---

celdaProbabilityMap

Renders probability and relative expression heatmaps to visualize the relationship between feature modules and cell populations.

Description

It is often useful to visualize to what degree each feature influences each cell cluster. This can also be useful for identifying features which may be redundant or unassociated with cell clustering.

Usage

```
celdaProbabilityMap(counts, celdaMod, ...)
```

Arguments

counts	Integer matrix. Rows represent features and columns represent cells. This matrix should be the same as the one used to generate 'celdaMod'.
celdaMod	Celda object of class "celda_C" or "celda_CG".
...	Additional parameters.

Value

A grob containing the specified plots

Examples

```
data(celdaCGSim, celdaCGMod)
celdaProbabilityMap(celdaCGSim$counts, celdaCGMod)
```

---

celdaProbabilityMap,celda_C-method

Probability map for a celda_C model

Description

Renders probability and relative expression heatmaps to visualize the relationship between cell populations and samples.

Usage

```
## S4 method for signature 'celda_C'
celdaProbabilityMap(counts, celdaMod,
  level = c("sample"), ...)
```

Arguments

counts	Integer matrix. Rows represent features and columns represent cells. This matrix should be the same as the one used to generate 'celdaMod'.
celdaMod	Celda object of class 'celda_C'.
level	Character. 'sample' will display the absolute probabilities and relative normalized abundance of each cell population in each sample. Default 'sample'.
...	Additional parameters.

Value

A grob containing the specified plots

See Also

'celda_C()' for clustering cells

Examples

```
data(celdaCSim, celdaCMod)
celdaProbabilityMap(celdaCSim$counts, celdaCMod)
```

`celdaProbabilityMap,celda_CG-method`
Probability map for a celda_CG model

Description

Renders probability and relative expression heatmaps to visualize the relationship between features and cell populations or cell populations and samples.

Usage

```
## S4 method for signature 'celda_CG'
celdaProbabilityMap(counts, celdaMod,
  level = c("cellPopulation", "sample"), ...)
```

Arguments

`counts`	Integer matrix. Rows represent features and columns represent cells. This matrix should be the same as the one used to generate 'celdaMod'.
`celdaMod`	Celda object of class 'celda_CG'.
`level`	Character. One of 'cellPopulation' or 'sample'. 'cellPopulation' will display the absolute probabilities and relative normalized expression of each module in each cell population. 'sample' will display the absolute probabilities and relative normalized abundance of each cell population in each sample. Default 'cellPopulation'.
`...`	Additional parameters.

Value

A grob containing the specified plots

See Also

'celda_CG()' for clustering features and cells

Examples

```
data(celdaCGSim, celdaCGMod)
celdaProbabilityMap(celdaCGSim$counts, celdaCGMod)
```

`celdaTsne` *Embeds cells in two dimensions using tSNE based on celda_CG results.*

Description

Embeds cells in two dimensions using tSNE based on celda_CG results.

Usage

```
celdaTsne(counts, celdaMod, maxCells = 25000, minClusterSize = 100,
  initialDims = 20, modules = NULL, perplexity = 20,
  maxIter = 2500, ...)
```

Arguments

`counts`	Integer matrix. Rows represent features and columns represent cells. This matrix should be the same as the one used to generate 'celdaMod'.
`celdaMod`	Celda object of class 'celda_CG'.
`maxCells`	Integer. Maximum number of cells to plot. Cells will be randomly subsampled if ncol(counts) > maxCells. Larger numbers of cells requires more memory. Default 25000.
`minClusterSize`	Integer. Do not subsample cell clusters below this threshold. Default 100.
`initialDims`	integer. The number of dimensions that should be retained in the initial PCA step. Default 20.
`modules`	Integer vector. Determines which features modules to use for tSNE. If NULL, all modules will be used. Default NULL.
`perplexity`	Numeric. Perplexity parameter for tSNE. Default 20.
`maxIter`	Integer. Maximum number of iterations in tSNE generation. Default 2500.
`...`	Additional parameters.

Value

Numeric Matrix of dimension 'ncol(counts)' x 2, colums representing the "X" and "Y" coordinates in the data's t-SNE represetation.

Examples

```
data(celdaCGSim, celdaCGMod)
tsneRes <- celdaTsne(celdaCGSim$counts, celdaCGMod)
```

`celdaTsne,celda_C-method` *tSNE for celda_C*

Description

Embeds cells in two dimensions using tSNE based on a 'celda_C' model. PCA on the normalized counts is used to reduce the number of features before applying tSNE.

Usage

```
## S4 method for signature 'celda_C'
celdaTsne(counts, celdaMod, maxCells = 25000,
  minClusterSize = 100, initialDims = 20, modules = NULL,
  perplexity = 20, maxIter = 2500, seed = 12345)
```

Arguments

`counts`	Integer matrix. Rows represent features and columns represent cells. This matrix should be the same as the one used to generate 'celdaMod'.
`celdaMod`	Celda object of class 'celda_C'.
`maxCells`	Integer. Maximum number of cells to plot. Cells will be randomly subsampled if ncol(counts) > maxCells. Larger numbers of cells requires more memory. Default 25000.
`minClusterSize`	Integer. Do not subsample cell clusters below this threshold. Default 100.
`initialDims`	Integer. PCA will be used to reduce the dimentionality of the dataset. The top 'initialDims' principal components will be used for tSNE. Default 20.
`modules`	Integer vector. Determines which features modules to use for tSNE. If NULL, all modules will be used. Default NULL.
`perplexity`	Numeric. Perplexity parameter for tSNE. Default 20.
`maxIter`	Integer. Maximum number of iterations in tSNE generation. Default 2500.
`seed`	Integer. Passed to with_seed. For reproducibility, a default value of 12345 is used. If NULL, no calls to with_seed are made.

Value

A two column matrix of t-SNE coordinates

See Also

'celda_C()' for clustering cells and 'celdaHeatmap()' for displaying expression

Examples

```
data(celdaCSim, celdaCMod)
tsneRes <- celdaTsne(celdaCSim$counts, celdaCMod)
```

`celdaTsne,celda_CG-method` *tSNE for celda_CG*

Description

Embeds cells in two dimensions using tSNE based on a 'celda_CG' model. tSNE is run on module probabilities to reduce the number of features instead of using PCA. Module probabilities square-root trasformed before applying tSNE.

Usage

```
## S4 method for signature 'celda_CG'
celdaTsne(counts, celdaMod, maxCells = 25000,
  minClusterSize = 100, initialDims = 20, modules = NULL,
  perplexity = 20, maxIter = 2500, seed = 12345)
```

Arguments

`counts`	Integer matrix. Rows represent features and columns represent cells. This matrix should be the same as the one used to generate 'celdaMod'.
`celdaMod`	Celda object of class 'celda_CG'.
`maxCells`	Integer. Maximum number of cells to plot. Cells will be randomly subsampled if ncol(counts) > maxCells. Larger numbers of cells requires more memory. Default 25000.
`minClusterSize`	Integer. Do not subsample cell clusters below this threshold. Default 100.
`initialDims`	Integer. PCA will be used to reduce the dimentionality of the dataset. The top 'initialDims' principal components will be used for tSNE. Default 20.
`modules`	Integer vector. Determines which features modules to use for tSNE. If NULL, all modules will be used. Default NULL.
`perplexity`	Numeric. Perplexity parameter for tSNE. Default 20.
`maxIter`	Integer. Maximum number of iterations in tSNE generation. Default 2500.
`seed`	Integer. Passed to with_seed. For reproducibility, a default value of 12345 is used. If NULL, no calls to with_seed are made.

Value

A two column matrix of t-SNE coordinates

See Also

'celda_CG()' for clustering features and cells and 'celdaHeatmap()' for displaying expression

Examples

```
data(celdaCGSim, celdaCGMod)
tsneRes <- celdaTsne(celdaCGSim$counts, celdaCGMod)
```

`celdaTsne,celda_G-method` *tSNE for celda_G*

Description

Embeds cells in two dimensions using tSNE based on a 'celda_G' model. tSNE is run on module probabilities to reduce the number of features instead of using PCA. Module probabilities square-root trasformed before applying tSNE.

Usage

```
## S4 method for signature 'celda_G'
celdaTsne(counts, celdaMod, maxCells = 25000,
  minClusterSize = 100, initialDims = 20, modules = NULL,
  perplexity = 20, maxIter = 2500, seed = 12345)
```

Arguments

`counts`	Integer matrix. Rows represent features and columns represent cells. This matrix should be the same as the one used to generate 'celdaMod'.
`celdaMod`	Celda object of class 'celda_G'.
`maxCells`	Integer. Maximum number of cells to plot. Cells will be randomly subsampled if ncol(conts) > maxCells. Larger numbers of cells requires more memory. Default 10000.
`minClusterSize`	Integer. Do not subsample cell clusters below this threshold. Default 100.
`initialDims`	Integer. PCA will be used to reduce the dimentionality of the dataset. The top 'initialDims' principal components will be used for tSNE. Default 20.
`modules`	Integer vector. Determines which feature modules to use for tSNE. If NULL, all modules will be used. Default NULL.
`perplexity`	Numeric. Perplexity parameter for tSNE. Default 20.
`maxIter`	Integer. Maximum number of iterations in tSNE generation. Default 2500.
`seed`	Integer. Passed to with_seed. For reproducibility, a default value of 12345 is used. If NULL, no calls to with_seed are made.

Value

A two column matrix of t-SNE coordinates.

See Also

'celda_G()' for clustering features and 'celdaHeatmap()' for displaying expression

Examples

```
data(celdaGSim, celdaGMod)
tsneRes <- celdaTsne(celdaGSim$counts, celdaGMod)
```

`celdaUmap` *Embeds cells in two dimensions using umap.*

Description

Embeds cells in two dimensions using umap.

Usage

```
celdaUmap(counts, celdaMod, maxCells = 25000, minClusterSize = 100,
  initialDims = 20, modules = NULL, seed = 12345,
  umapConfig = umap::umap.defaults)
```

Arguments

`counts`	Integer matrix. Rows represent features and columns represent cells. This matrix should be the same as the one used to generate 'celdaMod'.
`celdaMod`	Celda object of class 'celda_CG'.
`maxCells`	Integer. Maximum number of cells to plot. Cells will be randomly subsampled if ncol(counts) > maxCells. Larger numbers of cells requires more memory. Default 25000.
`minClusterSize`	Integer. Do not subsample cell clusters below this threshold. Default 100.
`initialDims`	Integer. PCA will be used to reduce the dimentionality of the dataset. The top 'initialDims' principal components will be used for umap. Default 20.
`modules`	Integer vector. Determines which features modules to use for tSNE. If NULL, all modules will be used. Default NULL.
`seed`	Integer. Passed to with_seed. For reproducibility, a default value of 12345 is used. If NULL, no calls to with_seed are made.
`umapConfig`	An object of class "umapConfig" specifying parameters to the UMAP algorithm.

Value

Numeric Matrix of dimension 'ncol(counts)' x 2, colums representing the "X" and "Y" coordinates in the data's t-SNE represetation.

Examples

```
data(celdaCGSim, celdaCGMod)
tsneRes <- celdaUmap(celdaCGSim$counts, celdaCGMod)
```

`celdaUmap,celda_C-method` *umap for celda_C*

Description

Embeds cells in two dimensions using umap based on a 'celda_C' model. PCA on the normalized counts is used to reduce the number of features before applying umap.

Usage

```
## S4 method for signature 'celda_C'
celdaUmap(counts, celdaMod, maxCells = 25000,
  minClusterSize = 100, modules = NULL, seed = 12345,
  umapConfig = umap::umap.defaults)
```

Arguments

`counts`	Integer matrix. Rows represent features and columns represent cells. This matrix should be the same as the one used to generate 'celdaMod'.
`celdaMod`	Celda object of class 'celda_C'.
`maxCells`	Integer. Maximum number of cells to plot. Cells will be randomly subsampled if ncol(counts) > maxCells. Larger numbers of cells requires more memory. Default 25000.
`minClusterSize`	Integer. Do not subsample cell clusters below this threshold. Default 100.
`modules`	Integer vector. Determines which features modules to use for UMAP. If NULL, all modules will be used. Default NULL.
`seed`	Integer. Passed to with_seed. For reproducibility, a default value of 12345 is used. If NULL, no calls to with_seed are made.
`umapConfig`	An object of class "umap.config" specifying parameters to the UMAP algorithm.

Value

A two column matrix of umap coordinates

See Also

'celda_C()' for clustering cells and 'celdaHeatmap()' for displaying expression.

Examples

```
data(celdaCSim, celdaCMod)
umapRes <- celdaUmap(celdaCSim$counts, celdaCMod)
```

`celdaUmap,celda_CG-method` *umap for celda_CG*

Description

Embeds cells in two dimensions using umap based on a 'celda_CG' model. umap is run on module probabilities to reduce the number of features instead of using PCA. Module probabilities square-root trasformed before applying tSNE.

Usage

```
## S4 method for signature 'celda_CG'
celdaUmap(counts, celdaMod, maxCells = 25000,
  minClusterSize = 100, modules = NULL, seed = 12345,
  umapConfig = umap::umap.defaults)
```

Arguments

`counts`	Integer matrix. Rows represent features and columns represent cells. This matrix should be the same as the one used to generate 'celdaMod'.
`celdaMod`	Celda object of class 'celda_CG'.
`maxCells`	Integer. Maximum number of cells to plot. Cells will be randomly subsampled if ncol(counts) > maxCells. Larger numbers of cells requires more memory. Default 25000.
`minClusterSize`	Integer. Do not subsample cell clusters below this threshold. Default 100.
`modules`	Integer vector. Determines which features modules to use for tSNE. If NULL, all modules will be used. Default NULL.
`seed`	Integer. Passed to with_seed. For reproducibility, a default value of 12345 is used. If NULL, no calls to with_seed are made.
`umapConfig`	Object of class 'umap.config'. Configures parameters for umap. Default 'umap::umap.defaults'.

Value

A two column matrix of umap coordinates

See Also

'celda_CG()' for clustering features and cells and 'celdaHeatmap()' for displaying expression.

Examples

```
data(celdaCGSim, celdaCGMod)
umapRes <- celdaUmap(celdaCGSim$counts, celdaCGMod)
```

`celdaUmap,celda_G-method` *umap for celda_G*

Description

Embeds cells in two dimensions using umap based on a 'celda_G' model. umap is run on module probabilities to reduce the number of features instead of using PCA. Module probabilities square-root trasformed before applying tSNE.

Usage

```
## S4 method for signature 'celda_G'
celdaUmap(counts, celdaMod, maxCells = 25000,
  minClusterSize = 100, modules = NULL, seed = 12345,
  umapConfig = umap::umap.defaults)
```

Arguments

`counts`	Integer matrix. Rows represent features and columns represent cells. This matrix should be the same as the one used to generate 'celdaMod'.
`celdaMod`	Celda object of class 'celda_CG'.
`maxCells`	Integer. Maximum number of cells to plot. Cells will be randomly subsampled if ncol(counts) > maxCells. Larger numbers of cells requires more memory. Default 25000.
`minClusterSize`	Integer. Do not subsample cell clusters below this threshold. Default 100.
`modules`	Integer vector. Determines which features modules to use for tSNE. If NULL, all modules will be used. Default NULL.
`seed`	Integer. Passed to with_seed. For reproducibility, a default value of 12345 is used. If NULL, no calls to with_seed are made.
`umapConfig`	Object of class 'umap.config'. Configures parameters for umap. Default 'umap::umap.defaults'.

Value

A two column matrix of umap coordinates

See Also

'celda_G()' for clustering features and cells and 'celdaHeatmap()' for displaying expression

Examples

```
data(celdaGSim, celdaGMod)
umapRes <- celdaUmap(celdaGSim$counts, celdaGMod)
```

celda_C *Cell clustering with Celda*

Description

Clusters the columns of a count matrix containing single-cell data into K subpopulations.

Usage

```
celda_C(counts, sampleLabel = NULL, K, alpha = 1, beta = 1,
  algorithm = c("EM", "Gibbs"), stopIter = 10, maxIter = 200,
  splitOnIter = 10, splitOnLast = TRUE, seed = 12345, nchains = 3,
  zInitialize = c("split", "random", "predefined"),
  countChecksum = NULL, zInit = NULL, logfile = NULL,
  verbose = TRUE)
```

Arguments

counts	Integer matrix. Rows represent features and columns represent cells.
sampleLabel	Vector or factor. Denotes the sample label for each cell (column) in the count matrix.
K	Integer. Number of cell populations.
alpha	Numeric. Concentration parameter for Theta. Adds a pseudocount to each cell population in each sample. Default 1.
beta	Numeric. Concentration parameter for Phi. Adds a pseudocount to each feature in each cell population. Default 1.
algorithm	String. Algorithm to use for clustering cell subpopulations. One of 'EM' or 'Gibbs'. The EM algorithm is faster, especially for larger numbers of cells. However, more chains may be required to ensure a good solution is found. If 'EM' is selected, then 'stopIter' will be automatically set to 1. Default 'EM'.
stopIter	Integer. Number of iterations without improvement in the log likelihood to stop inference. Default 10.
maxIter	Integer. Maximum number of iterations of Gibbs sampling or EM to perform. Default 200.
splitOnIter	Integer. On every 'splitOnIter' iteration, a heuristic will be applied to determine if a cell population should be reassigned and another cell population should be split into two clusters. To disable splitting, set to -1. Default 10.
splitOnLast	Integer. After 'stopIter' iterations have been performed without improvement, a heuristic will be applied to determine if a cell population should be reassigned and another cell population should be split into two clusters. If a split occurs, then 'stopIter' will be reset. Default TRUE.
seed	Integer. Passed to with_seed. For reproducibility, a default value of 12345 is used. If NULL, no calls to with_seed are made.
nchains	Integer. Number of random cluster initializations. Default 3.
zInitialize	Chararacter. One of 'random', 'split', or 'predefined'. With 'random', cells are randomly assigned to a populations. With 'split', cells will be split into sqrt(K) populations and then each popluation will be subsequently split into another sqrt(K) populations. With 'predefined', values in 'zInit' will be used to initialize 'z'. Default 'split'.

countChecksum	"Character. An MD5 checksum for the ‘counts‘ matrix. Default NULL.
zInit	Integer vector. Sets initial starting values of z. If NULL, starting values for each cell will be randomly sampled from ‘1:K‘. ’zInit’ can only be used when ‘initialize = ’random’‘. Default NULL.
logfile	Character. Messages will be redirected to a file named ‘logfile‘. If NULL, messages will be printed to stdout. Default NULL.
verbose	Logical. Whether to print log messages. Default TRUE.

Value

An object of class ‘celda_C‘ with the cell population clusters stored in in ‘z‘.

See Also

‘celda_G()‘ for feature clustering and ‘celda_CG()‘ for simultaneous clustering of features and cells. ‘celdaGridSearch()‘ can be used to run multiple values of K and multiple chains in parallel.

Examples

```
data(celdaCSim)
celdaCMod <- celda_C(celdaCSim$counts,
    K = celdaCSim$K,
    sampleLabel = celdaCSim$sampleLabel)
```

---

celda_CG *Cell and feature clustering with Celda*

Description

Clusters the rows and columns of a count matrix containing single-cell data into L modules and K subpopulations, respectively.

Usage

```
celda_CG(counts, sampleLabel = NULL, K, L, alpha = 1, beta = 1,
  delta = 1, gamma = 1, algorithm = c("EM", "Gibbs"),
  stopIter = 10, maxIter = 200, splitOnIter = 10,
  splitOnLast = TRUE, seed = 12345, nchains = 3,
  zInitialize = c("split", "random", "predefined"),
  yInitialize = c("split", "random", "predefined"),
  countChecksum = NULL, zInit = NULL, yInit = NULL, logfile = NULL,
  verbose = TRUE)
```

Arguments

counts	Integer matrix. Rows represent features and columns represent cells.
sampleLabel	Vector or factor. Denotes the sample label for each cell (column) in the count matrix.
K	Integer. Number of cell populations.
L	Integer. Number of feature modules.

alpha	Numeric. Concentration parameter for Theta. Adds a pseudocount to each cell population in each sample. Default 1.
beta	Numeric. Concentration parameter for Phi. Adds a pseudocount to each feature module in each cell population. Default 1.
delta	Numeric. Concentration parameter for Psi. Adds a pseudocount to each feature in each module. Default 1.
gamma	Numeric. Concentration parameter for Eta. Adds a pseudocount to the number of features in each module. Default 1.
algorithm	String. Algorithm to use for clustering cell subpopulations. One of 'EM' or 'Gibbs'. The EM algorithm for cell clustering is faster, especially for larger numbers of cells. However, more chains may be required to ensure a good solution is found. Default 'EM'.
stopIter	Integer. Number of iterations without improvement in the log likelihood to stop inference. Default 10.
maxIter	Integer. Maximum number of iterations of Gibbs sampling to perform. Default 200.
splitOnIter	Integer. On every 'splitOnIter' iteration, a heuristic will be applied to determine if a cell population or feature module should be reassigned and another cell population or feature module should be split into two clusters. To disable splitting, set to -1. Default 10.
splitOnLast	Integer. After 'stopIter' iterations have been performed without improvement, a heuristic will be applied to determine if a cell population or feature module should be reassigned and another cell population or feature module should be split into two clusters. If a split occurs, then 'stopIter' will be reset. Default TRUE.
seed	Integer. Passed to with_seed. For reproducibility, a default value of 12345 is used. If NULL, no calls to with_seed are made.
nchains	Integer. Number of random cluster initializations. Default 3.
zInitialize	Chararacter. One of 'random', 'split', or 'predefined'. With 'random', cells are randomly assigned to a populations. With 'split', cells will be split into sqrt(K) populations and then each popluation will be subsequently split into another sqrt(K) populations. With 'predefined', values in 'zInit' will be used to initialize 'z'. Default 'split'.
yInitialize	Chararacter. One of 'random', 'split', or 'predefined'. With 'random', features are randomly assigned to a modules. With 'split', features will be split into sqrt(L) modules and then each module will be subsequently split into another sqrt(L) modules. With 'predefined', values in 'yInit' will be used to initialize 'y'. Default 'split'.
countChecksum	Character. An MD5 checksum for the 'counts' matrix. Default NULL.
zInit	Integer vector. Sets initial starting values of z. If NULL, starting values for each cell will be randomly sampled from 1:K. 'zInit' can only be used when 'initialize' = 'random'. Default NULL.
yInit	Integer vector. Sets initial starting values of y. If NULL, starting values for each feature will be randomly sampled from 1:L. 'yInit' can only be used when 'initialize = 'random'. Default NULL.
logfile	Character. Messages will be redirected to a file named 'logfile'. If NULL, messages will be printed to stdout. Default NULL.
verbose	Logical. Whether to print log messages. Default TRUE.

Value

An object of class ‘celda_CG‘ with the cell populations clusters stored in in ‘z‘ and feature module clusters stored in ‘y‘.

See Also

‘celda_G()‘ for feature clustering and ‘celda_C()‘ for clustering cells. ‘celdaGridSearch()‘ can be used to run multiple values of K/L and multiple chains in parallel.

Examples

```
data(celdaCGSim)
celdaMod <- celda_CG(celdaCGSim$counts,
    K = celdaCGSim$K,
    L = celdaCGSim$L,
    sampleLabel = celdaCGSim$sampleLabel,
    nchains = 1)
```

---

celda_G *Feature clustering with Celda*

Description

Clusters the rows of a count matrix containing single-cell data into L modules.

Usage

```
celda_G(counts, L, beta = 1, delta = 1, gamma = 1, stopIter = 10,
  maxIter = 200, splitOnIter = 10, splitOnLast = TRUE,
  seed = 12345, nchains = 3, yInitialize = c("split", "random",
  "predefined"), countChecksum = NULL, yInit = NULL, logfile = NULL,
  verbose = TRUE)
```

Arguments

counts	Integer matrix. Rows represent features and columns represent cells.
L	Integer. Number of feature modules.
beta	Numeric. Concentration parameter for Phi. Adds a pseudocount to each feature module in each cell. Default 1.
delta	Numeric. Concentration parameter for Psi. Adds a pseudocount to each feature in each module. Default 1.
gamma	Numeric. Concentration parameter for Eta. Adds a pseudocount to the number of features in each module. Default 1.
stopIter	Integer. Number of iterations without improvement in the log likelihood to stop inference. Default 10.
maxIter	Integer. Maximum number of iterations of Gibbs sampling to perform. Default 200.
splitOnIter	Integer. On every ‘splitOnIter‘ iteration, a heuristic will be applied to determine if a feature module should be reassigned and another feature module should be split into two clusters. To disable splitting, set to -1. Default 10.

splitOnLast	Integer. After 'stopIter' iterations have been performed without improvement, a heuristic will be applied to determine if a cell population should be reassigned and another cell population should be split into two clusters. If a split occurs, then 'stopIter' will be reset. Default TRUE.
seed	Integer. Passed to with_seed. For reproducibility, a default value of 12345 is used. If NULL, no calls to with_seed are made.
nchains	Integer. Number of random cluster initializations. Default 3.
yInitialize	Chararacter. One of 'random', 'split', or 'predefined'. With 'random', features are randomly assigned to a modules. With 'split', features will be split into sqrt(L) modules and then each module will be subsequently split into another sqrt(L) modules. With 'predefined', values in 'yInit' will be used to initialize 'y'. Default 'split'.
countChecksum	Character. An MD5 checksum for the 'counts' matrix. Default NULL.
yInit	Integer vector. Sets initial starting values of y. If NULL, starting values for each feature will be randomly sampled from '1:L'. 'yInit' can only be used when 'initialize = 'random''. Default NULL.
logfile	Character. Messages will be redirected to a file named 'logfile'. If NULL, messages will be printed to stdout. Default NULL.
verbose	Logical. Whether to print log messages. Default TRUE.

Value

An object of class 'celda_G' with the feature module clusters stored in 'y'.

See Also

'celda_C()' for cell clustering and 'celda_CG()' for simultaneous clustering of features and cells. 'celdaGridSearch()' can be used to run multiple values of L and multiple chains in parallel.

Examples

```
data(celdaGSim)
celdaMod <- celda_G(celdaGSim$counts, L = celdaGSim$L)
```

---

clusterProbability *Get cluster probability*

Description

Get the probability of the cluster assignments generated during a celda run.

Usage

```
clusterProbability(counts, celdaMod, log = FALSE, ...)
```

Arguments

counts	Integer matrix. Rows represent features and columns represent cells. This matrix should be the same as the one used to generate 'celdaMod'.
celdaMod	celdaModel. Options available in 'celda::availableModels'.
log	Logical. If FALSE, then the normalized conditional probabilities will be returned. If TRUE, then the unnormalized log probabilities will be returned. Default FALSE.
...	Additional parameters.

Value

A numeric vector of the cluster assignment probabilties

Examples

```
data(celdaCGSim, celdaCGMod)
clusterProb <- clusterProbability(celdaCGSim$counts, celdaCGMod)
```

---

clusterProbability,celda_C-method

Conditional probabilities for cells in subpopulations from a Celda_C model

---

Description

Calculates the conditional probability of each cell belonging to each subpopulation given all other cell cluster assignments in a 'celda_C()' result.

Usage

```
## S4 method for signature 'celda_C'
clusterProbability(counts, celdaMod, log = FALSE,
  ...)
```

Arguments

counts	Integer matrix. Rows represent features and columns represent cells. This matrix should be the same as the one used to generate 'celdaMod'.
celdaMod	Celda object of class 'celda_C'.
log	Logical. If FALSE, then the normalized conditional probabilities will be returned. If TRUE, then the unnormalized log probabilities will be returned. Default FALSE.
...	Additional parameters.

Value

A list containging a matrix for the conditional cell subpopulation cluster probabilities.

See Also

'celda_C()' for clustering cells

Examples

```
data(celdaCSim, celdaCMod)
clusterProb <- clusterProbability(celdaCSim$counts, celdaCMod)
```

---

clusterProbability,celda_CG-method

Conditional probabilities for cells and features from a Celda_CG model

---

Description

Calculates the conditional probability of each cell belonging to each subpopulation given all other cell cluster assignments as well as each feature belonging to each module given all other feature cluster assignments in a 'celda_CG()' result.

Usage

```
## S4 method for signature 'celda_CG'
clusterProbability(counts, celdaMod, log = FALSE,
  ...)
```

Arguments

counts	Integer matrix. Rows represent features and columns represent cells. This matrix should be the same as the one used to generate 'celdaMod'.
celdaMod	Celda object of class 'celda_CG'.
log	Logical. If FALSE, then the normalized conditional probabilities will be returned. If TRUE, then the unnormalized log probabilities will be returned. Default FALSE.
...	Additional parameters.

Value

A list containging a matrix for the conditional cell and feature cluster probabilities.

See Also

'celda_CG()' for clustering features and cells

Examples

```
data(celdaCGSim, celdaCGMod)
clusterProb <- clusterProbability(celdaCGSim$counts, celdaCGMod)
```

clusterProbability,celda_G-method *Conditional probabilities for features in modules from a Celda_G model*

Description

Calculates the conditional probability of each feature belonging to each module given all other feature cluster assignments in a 'celda_G()' result.

Usage

```
## S4 method for signature 'celda_G'
clusterProbability(counts, celdaMod, log = FALSE,
  ...)
```

Arguments

counts	Integer matrix. Rows represent features and columns represent cells. This matrix should be the same as the one used to generate 'celdaMod'.
celdaMod	Celda object of class 'celda_G'.
log	Logical. If FALSE, then the normalized conditional probabilities will be returned. If TRUE, then the unnormalized log probabilities will be returned. Default FALSE.
...	Additional parameters.

Value

A list containging a matrix for the conditional cell cluster probabilities.

See Also

'celda_G()' for clustering features

Examples

```
data(celdaGSim, celdaGMod)
clusterProb <- clusterProbability(celdaGSim$counts, celdaGMod)
```

clusters *Get clustering outcomes from a celdaModel*

Description

Returns the z / y results corresponding to the cell / gene cluster labels determined by the provided celdaModel.

Usage

```
clusters(celdaMod)
```

Arguments

celdaMod	celdaModel. Options available in 'celda::availableModels'.

Value

List. Contains z (for celda_C and celdaCGModels) and/or y (for celda_G and celdaCGModels)

Examples

```
data(celdaCGMod)
clusters(celdaCGMod)
```

---

clusters,celdaModel-method

Get clustering outcomes from a celdaModel

Description

Returns the z / y results corresponding to the cell / gene cluster labels determined by the provided celdaModel.

Usage

```
## S4 method for signature 'celdaModel'
clusters(celdaMod)
```

Arguments

celdaMod	celdaModel. Options available in 'celda::availableModels'.

Value

List. Contains z (for celda_C and celdaCGModels) and/or y (for celda_G and celdaCGModels)

Examples

```
data(celdaCGMod)
clusters(celdaCGMod)
```

compareCountMatrix *Check count matrix consistency*

Description

Checks if the counts matrix is the same one used to generate the celda model object by comparing dimensions and MD5 checksum.

Usage

```
compareCountMatrix(counts, celdaMod, errorOnMismatch = TRUE)
```

Arguments

counts	Integer matrix. Rows represent features and columns represent cells.
celdaMod	Celda model object.
errorOnMismatch	Logical. Whether to throw an error in the event of a mismatch. Default TRUE.

Value

Returns TRUE if provided count matrix matches the one used in the celda object and/or 'errorOnMismatch = FALSE', FALSE otherwise.

Examples

```
data(celdaCGSim, celdaCGMod)
compareCountMatrix(celdaCGSim$counts, celdaCGMod, errorOnMismatch = FALSE)
```

contaminationSim *contaminationSim*

Description

Generated by simulateContaminatedMatrix

Usage

```
contaminationSim
```

Format

A list

Details

A toy contamination data generated by simulateContaminatedMatrix

countChecksum *Get the MD5 hash of the count matrix from the celdaList*

Description

Returns the MD5 hash of the count matrix used to generate the celdaList.

Usage

```
countChecksum(celdaList)
```

Arguments

celdaList	An object of class celdaList.

Value

A character string of length 32 containing the MD5 digest of the count matrix.

Examples

```
data(celdaCGGridSearchRes)
countChecksum <- countChecksum(celdaCGGridSearchRes)
```

countChecksum,celdaList-method *Get the MD5 hash of the count matrix from the celdaList*

Description

Returns the MD5 hash of the count matrix used to generate the celdaList.

Usage

```
## S4 method for signature 'celdaList'
countChecksum(celdaList)
```

Arguments

celdaList	An object of class celdaList.

Value

A character string of length 32 containing the MD5 digest of the count matrix.

Examples

```
data(celdaCGGridSearchRes)
countChecksum <- countChecksum(celdaCGGridSearchRes)
```

`decontX` *Decontaminate count matrix*

Description

This function updates decontamination on dataset with multiple batches.

Usage

```
decontX(counts, z = NULL, batch = NULL, maxIter = 200, delta = 10,
  logfile = NULL, verbose = TRUE, seed = 12345)
```

Arguments

`counts`	Numeric/Integer matrix. Observed count matrix, rows represent features and columns represent cells.
`z`	Integer vector. Cell population labels. Default NULL.
`batch`	Integer vector. Cell batch labels. Default NULL.
`maxIter`	Integer. Maximum iterations of EM algorithm. Default to be 200.
`delta`	Numeric. Symmetric concentration parameter for Theta. Default to be 10.
`logfile`	Character. Messages will be redirected to a file named ‘logfile‘. If NULL, messages will be printed to stdout. Default NULL.
`verbose`	Logical. Whether to print log messages. Default TRUE.
`seed`	Integer. Passed to with_seed. For reproducibility, a default value of 12345 is used. If NULL, no calls to with_seed are made.

Value

A list object which contains the decontaminated count matrix and related parameters.

Examples

```
data(contaminationSim)
deconC <- decontX(
  counts = contaminationSim$rmat + contaminationSim$cmat,
  z = contaminationSim$z, maxIter = 3
)
deconBg <- decontX(
  counts = contaminationSim$rmat + contaminationSim$cmat,
  maxIter = 3
)
```

differentialExpression *Differential expression for cell subpopulations using MAST*

Description

Uses MAST to find differentially expressed features for specified cell subpopulations.

Usage

```
differentialExpression(counts, celdaMod, c1, c2 = NULL,
  onlyPos = FALSE, log2fcThreshold = NULL, fdrThreshold = 1)
```

Arguments

counts	Integer matrix. Rows represent features and columns represent cells. This matrix should be the same as the one used to generate 'celdaMod'.
celdaMod	Celda object of class 'celda_C' or 'celda_CG'.
c1	Integer vector. Cell populations to include in group 1 for the differential expression analysis.
c2	Integer vector. Cell populations to include in group 2 for the differential expression analysis. If NULL, the clusters in the c1 group are compared to all other clusters. Default NULL.
onlyPos	Logical. Whether to only return markers with positive log2 fold change. Default FALSE.
log2fcThreshold	Numeric. A number greater than 0 that specifies the absolute log2 fold change threshold. Only features with absolute value above this threshold will be returned. If NULL, this filter will not be applied. Default NULL.
fdrThreshold	Numeric. A number between 0 and 1 that specifies the false discovery rate (FDR) threshold. Only features below this threshold will be returned. Default 1.

Value

Data frame containing MAST results including statistics such as p-value, log2 fold change, and FDR.

Examples

```
data(celdaCGSim, celdaCGMod)
clusterDiffexpRes = differentialExpression(celdaCGSim$counts,
    celdaCGMod, c1 = c(1, 2))
```

distinctColors *Create a color palette*

Description

Generate a palette of ‘n‘ distinct colors.

Usage

```
distinctColors(n, hues = c("red", "cyan", "orange", "blue", "yellow",
  "purple", "green", "magenta"), saturationRange = c(0.7, 1),
  valueRange = c(0.7, 1))
```

Arguments

n	Integer. Number of colors to generate.
hues	Character vector. Colors available from ‘colors()‘. These will be used as the base colors for the clustering scheme in HSV. Different saturations and values will be generated for each hue. Default c("red", "cyan", "orange", "blue", "yellow", "purple", "green", "magenta").
saturationRange	Numeric vector. A vector of length 2 denoting the saturation for HSV. Values must be in [0,1]. Default: c(0.25, 1).
valueRange	Numeric vector. A vector of length 2 denoting the range of values for HSV. Values must be in [0,1]. Default: ‘c(0.5, 1)‘.

Value

A vector of distinct colors that have been converted to HEX from HSV.

Examples

```
colorPal <- distinctColors(6) # can be used in plotting functions
```

eigenMatMultInt *Fast matrix multiplication for double x int*

Description

Fast matrix multiplication for double x int

Usage

```
eigenMatMultInt(A, B)
```

Arguments

A	a double matrix
B	an integer matrix

Value

An integer matrix representing the product of A and B

`factorizeMatrix` *Generate factorized matrices showing each feature's influence on cell / gene clustering*

Description

Generate factorized matrices showing each feature's influence on cell / gene clustering

Usage

```
factorizeMatrix(counts, celdaMod, type = c("counts", "proportion",
  "posterior"))
```

Arguments

`counts`	Integer matrix. Rows represent features and columns represent cells. This matrix should be the same as the one used to generate 'celdaMod'.
`celdaMod`	Celda object of class "celda_C", "celda_G", or "celda_CG".
`type`	A character vector containing one or more of "counts", "proportions", or "posterior". "counts" returns the raw number of counts for each entry in each matrix. "proportions" returns the counts matrix where each vector is normalized to a probability distribution. "posterior" returns the posterior estimates which include the addition of the Dirichlet concentration parameter (essentially as a pseudocount).

Value

A list of lists of the types of factorized matrices specified

Examples

```
data(celdaCGSim, celdaCGMod)
factorizedMatrices <- factorizeMatrix(
    celdaCGSim$counts, celdaCGMod,
    "posterior"
)
```

factorizeMatrix,celda_C-method
Matrix factorization for results from celda_C()

Description

Generates factorized matrices showing the contribution of each feature in each cell population or each cell population in each sample.

Usage

```
## S4 method for signature 'celda_C'
factorizeMatrix(counts, celdaMod, type = c("counts",
  "proportion", "posterior"))
```

Arguments

counts	Integer matrix. Rows represent features and columns represent cells. This matrix should be the same as the one used to generate ‘celdaMod‘.
celdaMod	Celda object of class "celda_C".
type	Character vector. A vector containing one or more of "counts", "proportion", or "posterior". "counts" returns the raw number of counts for each factorized matrix. "proportions" returns the normalized probabilities for each factorized matrix, which are calculated by dividing the raw counts in each factorized matrix by the total counts in each column. "posterior" returns the posterior estimates. Default ‘c("counts", "proportion", "posterior")‘.

Value

A list with elements for ‘counts‘, ‘proportions‘, or ‘posterior‘ probabilities. Each element will be a list containing factorized matrices for ‘module‘ and ‘sample‘.

See Also

‘celda_C()‘ for clustering cells

Examples

```
data(celdaCSim, celdaCMod)
factorizedMatrices <- factorizeMatrix(celdaCSim$counts,
    celdaCMod, "posterior")
```

`factorizeMatrix,celda_CG-method`
Matrix factorization for results from celda_CG

Description

Generates factorized matrices showing the contribution of each feature in each module, each module in each cell and/or cell population, and each cell population in each sample.

Usage

```
## S4 method for signature 'celda_CG'
factorizeMatrix(counts, celdaMod, type = c("counts",
  "proportion", "posterior"))
```

Arguments

`counts`	Integer matrix. Rows represent features and columns represent cells. This matrix should be the same as the one used to generate 'celdaMod'.
`celdaMod`	Celda model. Options are "celda_C" or "celda_CG". Celda object of class "celda_CG".
`type`	Character vector. A vector containing one or more of "counts", "proportion", or "posterior". "counts" returns the raw number of counts for each factorized matrix. "proportions" returns the normalized probabilities for each factorized matrix, which are calculated by dividing the raw counts in each factorized matrix by the total counts in each column. "posterior" returns the posterior estimates. Default 'c("counts", "proportion", "posterior")'.

Value

A list with elements for 'counts', 'proportions', or 'posterior' probabilities. Each element will be a list containing factorized matrices for 'module', 'cellPopulation', and 'sample'. Additionally, the contribution of each module in each individual cell will be included in the 'cell' element of 'counts' and 'proportions' elements.

See Also

'celda_CG()' for clustering features and cells

Examples

```
data(celdaCGSim, celdaCGMod)
factorizedMatrices <- factorizeMatrix(celdaCGSim$counts,
    celdaCGMod, "posterior")
```

`factorizeMatrix,celda_G-method`
Matrix factorization for results from celda_G

Description

Generates factorized matrices showing the contribution of each feature in each module and each module in each cell.

Usage

```
## S4 method for signature 'celda_G'
factorizeMatrix(counts, celdaMod, type = c("counts",
  "proportion", "posterior"))
```

Arguments

`counts`	Integer matrix. Rows represent features and columns represent cells. This matrix should be the same as the one used to generate ‘celdaMod‘.
`celdaMod`	Celda object of class "celda_G".
`type`	Character vector. A vector containing one or more of "counts", "proportion", or "posterior". "counts" returns the raw number of counts for each factorized matrix. "proportions" returns the normalized probabilities for each factorized matrix, which are calculated by dividing the raw counts in each factorized matrix by the total counts in each column. "posterior" returns the posterior estimates. Default ‘c("counts", "proportion", "posterior")‘.

Value

A list with elements for ‘counts‘, ‘proportions‘, or ‘posterior‘ probabilities. Each element will be a list containing factorized matrices for ‘module‘ and ‘cell‘.

See Also

‘celda_G()‘ for clustering features

Examples

```
data(celdaGSim, celdaGMod)
factorizedMatrices <- factorizeMatrix(celdaGSim$counts,
    celdaGMod, "posterior")
```

fastNormProp *Fast normalization for numeric matrix*

Description

Fast normalization for numeric matrix

Usage

```
fastNormProp(R_counts, R_alpha)
```

Arguments

R_counts	An integer matrix
R_alpha	A double value to be added to the matrix as a pseudocount

Value

A numeric matrix where the columns have been normalized to proportions

fastNormPropLog *Fast normalization for numeric matrix*

Description

Fast normalization for numeric matrix

Usage

```
fastNormPropLog(R_counts, R_alpha)
```

Arguments

R_counts	An integer matrix
R_alpha	A double value to be added to the matrix as a pseudocount

Value

A numeric matrix where the columns have been normalized to proportions

fastNormPropSqrt *Fast normalization for numeric matrix*

Description

Fast normalization for numeric matrix

Usage

```
fastNormPropSqrt(R_counts, R_alpha)
```

Arguments

R_counts	An integer matrix
R_alpha	A double value to be added to the matrix as a pseudocount

Value

A numeric matrix where the columns have been normalized to proportions

featureModuleLookup *Obtain the gene module of a gene of interest*

Description

This function will output the corresponding feature module for a specified list of genes from a celdaModel.

Usage

```
featureModuleLookup(counts, celdaMod, feature, exactMatch = TRUE)
```

Arguments

counts	Integer matrix. Rows represent features and columns represent cells. This matrix should be the same as the one used to generate 'celdaMod'.
celdaMod	Model of class "celda_G" or "celda_CG".
feature	Character vector. Identify feature modules for the specified feature names.
exactMatch	Logical. Whether to look for exactMatch of the gene name within counts matrix. Default TRUE.

Value

List. Each entry corresponds to the feature module determined for the provided features

Examples

```
data(celdaCGSim, celdaCGMod)
featureModuleLookup(
    counts = celdaCGSim$counts,
    celdaMod = celdaCGMod, "Gene_1")
```

featureModuleLookup,celda_C-method

Lookup the module of a feature

Description

Finds the module assignments of given features in a 'celda_C()' model.

Usage

```
## S4 method for signature 'celda_C'
featureModuleLookup(counts, celdaMod, feature,
  exactMatch = TRUE)
```

Arguments

counts	Integer matrix. Rows represent features and columns represent cells. This matrix should be the same as the one used to generate 'celdaMod'.
celdaMod	Model of class 'celda_C'.
feature	Character vector. The module assignemnts will be found for feature names in this vector.
exactMatch	Logical. Whether an exact match or a partial match using 'grep()' is required to look up the feature in the rownames of the counts matrix. Default TRUE.

Value

List. Each element contains the module of the provided feature.

featureModuleLookup,celda_CG-method

Lookup the module of a feature

Description

Finds the module assignments of given features in a 'celda_G()' model

Usage

```
## S4 method for signature 'celda_CG'
featureModuleLookup(counts, celdaMod, feature,
  exactMatch = TRUE)
```

Arguments

counts	Integer matrix. Rows represent features and columns represent cells. This matrix should be the same as the one used to generate 'celdaMod'.
celdaMod	Model of class 'celda_CG'.
feature	Character vector. The module assignemnts will be found for feature names in this vector.
exactMatch	Logical. Whether an exact match or a partial match using 'grep()' is used to look up the feature in the rownames of the counts matrix. Default TRUE.

Value

List. Each element contains the module of the provided feature.

See Also

'celda_CG()' for clustering features and cells

Examples

```
data(celdaCGSim, celdaCGMod)
module <- featureModuleLookup(celdaCGSim$counts,
    celdaCGMod,
    c("Gene_1", "Gene_XXX"))
```

`featureModuleLookup,celda_G-method`

Lookup the module of a feature

Description

Finds the module assignments of given features in a 'celda_G()' model.

Usage

```
## S4 method for signature 'celda_G'
featureModuleLookup(counts, celdaMod, feature,
  exactMatch = TRUE)
```

Arguments

`counts`	Integer matrix. Rows represent features and columns represent cells. This matrix should be the same as the one used to generate 'celdaMod'.
`celdaMod`	Model of class 'celda_G'.
`feature`	Character vector. The module assignemnts will be found for feature names in this vector.
`exactMatch`	Logical. Whether an exact match or a partial match using 'grep()' is used to look up the feature in the rownames of the counts matrix. Default TRUE.

Value

List. Each element contains the module of the provided feature.

See Also

'celda_G()' for clustering features

Examples

```
data(celdaGSim, celdaGMod)
module <- featureModuleLookup(celdaGSim$counts,
    celdaGMod,
    c("Gene_1", "Gene_XXX"))
```

featureModuleTable *Outputting a feature module table*

Description

Creates a table that contains the list of features in each feature module.

Usage

```
featureModuleTable(counts, celdaMod, outputFile = NULL)
```

Arguments

counts	Integer matrix. Rows represent features and columns represent cells.
celdaMod	Celda object of class "celda_G" or "celda_CG".
outputFile	File name for feature module table. If NULL, file will not be created. Default NULL.

Value

Matrix. Contains a list of features per each column (feature module)

Examples

```
data(celdaCGSim, celdaCGMod)
featureModuleTable(celdaCGSim$counts, celdaCGMod, outputFile = NULL)
```

geneSetEnrich *Gene set enrichment*

Description

Identify and return significantly-enriched terms for each gene module in a Celda object. Performs gene set enrichment analysis for Celda identified modules using the enrichR package.

Usage

```
geneSetEnrich(counts, celdaModel, databases, fdr = 0.05)
```

Arguments

counts	Integer count matrix. Rows represent genes and columns represent cells. Row names of the matrix should be gene names.
celdaModel	Celda object of class ‘celda_G‘ or ‘celda_CG‘.
databases	Character vector. Name of reference database. Available databases can be viewed by running `enrichR::listEnrichrDbs()`.
fdr	False discovery rate (FDR). Numeric. Cutoff value for adjusted p-value, terms with FDR below this value are considered significantly enriched.

Value

List of length 'L' where each member contains the significantly enriched terms for the corresponding module.

Author(s)

Ahmed Youssef

Examples

```
library(M3DExampleData)
counts <- M3DExampleData::Mmus_example_list$data
#subset 100 genes for fast clustering
counts <- counts[1500:2000, ]
#cluster genes into 10 modules for quick demo
cm <- celda_G(counts = as.matrix(counts), L = 10, verbose = FALSE)
gse <- geneSetEnrich(counts,
    cm,
    databases = c('GO_Biological_Process_2018','GO_Molecular_Function_2018'))
```

---

logLikelihood *Calculate LogLikelihood*

Description

Calculate a log-likelihood for a user-provided cluster assignment and count matrix, per the desired celdaModel.

Usage

```
logLikelihood(counts, model, ...)
```

Arguments

counts	The counts matrix used to generate the provided cluster assignments.
model	celdaModel. Options available in 'celda::availableModels'.
...	Additional parameters.

Value

The log-likelihood of the provided cluster assignment for the provided counts matrix.

Examples

```
data(celdaCGSim)
loglik <- logLikelihood(celdaCGSim$counts,
    model = "celda_CG",
    sampleLabel = celdaCGSim$sampleLabel,
    z = celdaCGSim$z, y = celdaCGSim$y,
    K = celdaCGSim$K, L = celdaCGSim$L,
    alpha = celdaCGSim$alpha, beta = celdaCGSim$beta,
    gamma = celdaCGSim$gamma, delta = celdaCGSim$delta
)
```

`logLikelihoodcelda_C` *Calculate Celda_C log likelihood*

Description

Calculates the log likelihood for user-provided cell population clusters using the 'celda_C()' model.

Usage

```
logLikelihoodcelda_C(counts, sampleLabel, z, K, alpha, beta)
```

Arguments

`counts`	Integer matrix. Rows represent features and columns represent cells.
`sampleLabel`	Vector or factor. Denotes the sample label for each cell (column) in the count matrix.
`z`	Numeric vector. Denotes cell population labels.
`K`	Integer. Number of cell populations.
`alpha`	Numeric. Concentration parameter for Theta. Adds a pseudocount to each cell population in each sample. Default 1.
`beta`	Numeric. Concentration parameter for Phi. Adds a pseudocount to each feature in each cell population. Default 1.

Value

Numeric. The log likelihood for the given cluster assignments

See Also

'celda_C()' for clustering cells

Examples

```
data(celdaCSim)
loglik <- logLikelihoodcelda_C(celdaCSim$counts,
    sampleLabel = celdaCSim$sampleLabel,
    z = celdaCSim$z,
    K = celdaCSim$K,
    alpha = celdaCSim$alpha,
    beta = celdaCSim$beta)

loglik <- logLikelihood(celdaCSim$counts,
    model = "celda_C",
    sampleLabel = celdaCSim$sampleLabel,
    z = celdaCSim$z,
    K = celdaCSim$K,
    alpha = celdaCSim$alpha,
    beta = celdaCSim$beta)
```

`logLikelihoodcelda_CG` *Calculate Celda_CG log likelihood*

Description

Calculates the log likelihood for user-provided cell population and feature module clusters using the 'celda_CG()' model.

Usage

```
logLikelihoodcelda_CG(counts, sampleLabel, z, y, K, L, alpha, beta, delta,
  gamma)
```

Arguments

`counts`	Integer matrix. Rows represent features and columns represent cells.
`sampleLabel`	Vector or factor. Denotes the sample label for each cell (column) in the count matrix.
`z`	Numeric vector. Denotes cell population labels.
`y`	Numeric vector. Denotes feature module labels.
`K`	Integer. Number of cell populations.
`L`	Integer. Number of feature modules.
`alpha`	Numeric. Concentration parameter for Theta. Adds a pseudocount to each cell population in each sample. Default 1.
`beta`	Numeric. Concentration parameter for Phi. Adds a pseudocount to each feature module in each cell population. Default 1.
`delta`	Numeric. Concentration parameter for Psi. Adds a pseudocount to each feature in each module. Default 1.
`gamma`	Numeric. Concentration parameter for Eta. Adds a pseudocount to the number of features in each module. Default 1.

Value

The log likelihood for the given cluster assignments

See Also

'celda_CG()' for clustering features and cells

Examples

```
data(celdaCGSim)
loglik <- logLikelihoodcelda_CG(celdaCGSim$counts,
    sampleLabel = celdaCGSim$sampleLabel,
    z = celdaCGSim$z,
    y = celdaCGSim$y,
    K = celdaCGSim$K,
    L = celdaCGSim$L,
    alpha = celdaCGSim$alpha,
```

```
    beta = celdaCGSim$beta,
    gamma = celdaCGSim$gamma,
    delta = celdaCGSim$delta)

loglik <- logLikelihood(celdaCGSim$counts,
    model = "celda_CG",
    sampleLabel = celdaCGSim$sampleLabel,
    z = celdaCGSim$z,
    y = celdaCGSim$y,
    K = celdaCGSim$K,
    L = celdaCGSim$L,
    alpha = celdaCGSim$alpha,
    beta = celdaCGSim$beta,
    gamma = celdaCGSim$gamma,
    delta = celdaCGSim$delta)
```

`logLikelihoodcelda_G` *Calculate Celda_G log likelihood*

Description

Calculates the log likelihood for user-provided feature module clusters using the 'celda_G()' model.

Usage

```
logLikelihoodcelda_G(counts, y, L, beta, delta, gamma)
```

Arguments

`counts`	Integer matrix. Rows represent features and columns represent cells.
`y`	Numeric vector. Denotes feature module labels.
`L`	Integer. Number of feature modules.
`beta`	Numeric. Concentration parameter for Phi. Adds a pseudocount to each feature module in each cell. Default 1.
`delta`	Numeric. Concentration parameter for Psi. Adds a pseudocount to each feature in each module. Default 1.
`gamma`	Numeric. Concentration parameter for Eta. Adds a pseudocount to the number of features in each module. Default 1.

Value

The log-likelihood for the given cluster assignments.

See Also

'celda_G()' for clustering features

Examples

```
data(celdaGSim)
loglik <- logLikelihoodcelda_G(celdaGSim$counts,
    y = celdaGSim$y,
    L = celdaGSim$L,
    beta = celdaGSim$beta,
    delta = celdaGSim$delta,
    gamma = celdaGSim$gamma)

loglik <- logLikelihood(celdaGSim$counts,
    model = "celda_G",
    y = celdaGSim$y,
    L = celdaGSim$L,
    beta = celdaGSim$beta,
    delta = celdaGSim$delta,
    gamma = celdaGSim$gamma)
```

---

logLikelihoodHistory *Get log-likelihood history*

Description

Retrieves the complete log-likelihood from all iterations of Gibbs sampling used to generate a celdaModel.

Usage

```
logLikelihoodHistory(celdaMod)
```

Arguments

celdaMod	celdaModel. Options available in 'celda::availableModels'.

Value

Numeric. The log-likelihood at each step of Gibbs sampling used to generate the model.

Examples

```
data(celdaCGMod)
logLikelihoodHistory(celdaCGMod)
```

logLikelihoodHistory,celdaModel-method *Get log-likelihood history*

Description

Retrieves the complete log-likelihood from all iterations of Gibbs sampling used to generate a celdaModel.

Usage

```
## S4 method for signature 'celdaModel'
logLikelihoodHistory(celdaMod)
```

Arguments

celdaMod	celdaModel. Options available in 'celda::availableModels'.

Value

Numeric. The log-likelihood at each step of Gibbs sampling used to generate the model.

Examples

```
data(celdaCGMod)
logLikelihoodHistory(celdaCGMod)
```

matrixNames *Get feature, cell and sample names from a celdaModel*

Description

Retrieves the row, column, and sample names used to generate a celdaModel.

Usage

```
matrixNames(celdaMod)
```

Arguments

celdaMod	celdaModel. Options available in 'celda::availableModels'.

Value

List. Contains row, column, and sample character vectors corresponding to the values provided when the celdaModel was generated.

Examples

```
data(celdaCGMod)
matrixNames(celdaCGMod)
```

matrixNames,celdaModel-method
Get feature, cell and sample names from a celdaModel

Description

Retrieves the row, column, and sample names used to generate a celdaModel.

Usage

```
## S4 method for signature 'celdaModel'
matrixNames(celdaMod)
```

Arguments

celdaMod	celdaModel. Options available in 'celda::availableModels'.

Value

List. Contains row, column, and sample character vectors corresponding to the values provided when the celdaModel was generated.

Examples

```
data(celdaCGMod)
matrixNames(celdaCGMod)
```

moduleHeatmap *Heatmap for featureModules*

Description

Renders a heatmap for selected featureModules. Cells are ordered from those with the lowest probability of the module on the left to the highest probability on the right. If more than one module is used, then cells will be ordered by the probabilities of the first module only. Features are ordered from those with the highest probability in the module on the top to the lowest probability on the bottom.

Usage

```
moduleHeatmap(counts, celdaMod, featureModule = 1, topCells = 100,
  topFeatures = NULL, normalizedCounts = NA, scaleRow = scale,
  showFeaturenames = TRUE)
```

Arguments

counts	Integer matrix. Rows represent features and columns represent cells. This matrix should be the same as the one used to generate 'celdaMod'.
celdaMod	Celda object of class 'celda_G' or 'celda_CG'.
featureModule	Integer Vector. The featureModule(s) to display. Multiple modules can be included in a vector.
topCells	Integer. Number of cells with the highest and lowest probabilities for this module to include in the heatmap. For example, if 'topCells' = 50, the 50 cells with the lowest probability and the 50 cells with the highest probability for that featureModule will be included. If NULL, all cells will be plotted. Default 100.
topFeatures	Integer. Plot 'topFeatures' with the highest probability in the featureModule. If NULL, plot all features in the module. Default NULL.
normalizedCounts	Integer matrix. Rows represent features and columns represent cells. This matrix should correspond to the one provided for 'counts', but should be passed through. If NA, normalize 'counts'. Default NA. 'normalizeCounts(counts, "proportion", transformationFun=sqrt)'. Use of this parameter is particularly useful for plotting many moduleHeatmaps, where normalizing the counts matrix repeatedly would be too time consuming.
scaleRow	Character. Which function to use to scale each individual row. Set to NULL to disable. Occurs after normalization and log transformation. For example, 'scale' will Z-score transform each row. Default 'scale'.
showFeaturenames	Logical. Wheter feature names should be displayed. Default TRUE.

Value

A list containing row and column dendrograms as well as a gtable for grob plotting

Examples

```
data(celdaCGSim, celdaCGMod)
moduleHeatmap(celdaCGSim$counts, celdaCGMod)
```

---

normalizeCounts *Normalization of count data*

Description

Performs normalization, transformation, and/or scaling of a counts matrix

Usage

```
normalizeCounts(counts, normalize = c("proportion", "cpm", "median",
  "mean"), transformationFun = NULL, scaleFun = NULL,
  pseudocountNormalize = 0, pseudocountTransform = 0)
```

Arguments

counts	Integer matrix. Rows represent features and columns represent cells.
normalize	Character. Divides counts by the library sizes for each cell. One of 'proportion', 'cpm', 'median', or 'mean'. 'proportion' uses the total counts for each cell as the library size. 'cpm' divides the library size of each cell by one million to produce counts per million. 'median' divides the library size of each cell by the median library size across all cells. 'mean' divides the library size of each cell by the mean library size across all cells.
transformationFun	Function. Applys a transformation such as 'sqrt', 'log', 'log2', 'log10', or 'log1p'. If NULL, no transformation will be applied. Occurs after normalization. Default NULL.
scaleFun	Function. Scales the rows of the normalized and transformed count matrix. For example, 'scale' can be used to z-score normalize the rows. Default NULL.
pseudocountNormalize	Numeric. Add a pseudocount to counts before normalization. Default 0.
pseudocountTransform	Numeric. Add a pseudocount to normalized counts before applying the transformation function. Adding a pseudocount can be useful before applying a log transformation. Default 0.

Value

Numeric Matrix. A normalized matrix.

Examples

```
data(celdaCGSim)
normalizedCounts <- normalizeCounts(celdaCGSim$counts, "proportion",
    pseudocountNormalize = 1)
```

---

params *Get parameter values provided for celdaModel creation*

Description

Retrieves the K/L, model priors (e.g. alpha, beta), and count matrix checksum parameters provided during the creation of the provided celdaModel.

Usage

```
params(celdaMod)
```

Arguments

celdaMod	celdaModel. Options available in 'celda::availableModels'.

Value

List. Contains the model-specific parameters for the provided celda model object depending on its class.

Examples

```
data(celdaCGMod)
params(celdaCGMod)
```

`params,celdaModel-method` *Get parameter values provided for celdaModel creation*

Description

Retrieves the K/L, model priors (e.g. alpha, beta), and count matrix checksum parameters provided during the creation of the provided celdaModel.

Usage

```
## S4 method for signature 'celdaModel'
params(celdaMod)
```

Arguments

`celdaMod`	celdaModel. Options available in 'celda::availableModels'.

Value

List. Contains the model-specific parameters for the provided celda model object depending on its class.

Examples

```
data(celdaCGMod)
params(celdaCGMod)
```

`perplexity` *Calculate the perplexity from a single celdaModel*

Description

Perplexity can be seen as a measure of how well a provided set of cluster assignments fit the data being clustered.

Usage

```
perplexity(counts, celdaMod, newCounts = NULL)
```

Arguments

`counts`	Integer matrix. Rows represent features and columns represent cells. This matrix should be the same as the one used to generate 'celdaMod'.
`celdaMod`	celdaModel. Options available in 'celda::availableModels'.
`newCounts`	A newCounts matrix used to calculate perplexity. If NULL, perplexity will be calculated for the 'counts' matrix. Default NULL.

Value

Numeric. The perplexity for the provided count data and model.

Examples

```
data(celdaCGSim, celdaCGMod)
perplexity <- perplexity(celdaCGSim$counts, celdaCGMod)
```

`perplexity,celda_C-method`

Calculate the perplexity on new data with a celda_C model

Description

Perplexity is a statistical measure of how well a probability model can predict new data. Lower perplexity indicates a better model.

Usage

```
## S4 method for signature 'celda_C'
perplexity(counts, celdaMod, newCounts = NULL)
```

Arguments

`counts`	Integer matrix. Rows represent features and columns represent cells. This matrix should be the same as the one used to generate ‘celdaMod‘.
`celdaMod`	Celda object of class "celda_C"
`newCounts`	A new counts matrix used to calculate perplexity. If NULL, perplexity will be calculated for the ’counts’ matrix. Default NULL.

Value

Numeric. The perplexity for the provided count data and model.

See Also

‘celda_C()‘ for clustering cells

Examples

```
data(celdaCSim, celdaCMod)
perplexity <- perplexity(celdaCSim$counts, celdaCMod)
```

perplexity,celda_CG-method
Calculate the perplexity on new data with a celda_CG model

Description

Perplexity is a statistical measure of how well a probability model can predict new data. Lower perplexity indicates a better model.

Usage

```
## S4 method for signature 'celda_CG'
perplexity(counts, celdaMod, newCounts = NULL)
```

Arguments

counts	Integer matrix. Rows represent features and columns represent cells. This matrix should be the same as the one used to generate ‘celdaMod‘.
celdaMod	Celda object of class "celda_C", "celda_G" or "celda_CG".
newCounts	A new counts matrix used to calculate perplexity. If NULL, perplexity will be calculated for the ’counts’ matrix. Default NULL.

Value

Numeric. The perplexity for the provided count data and model.

See Also

‘celda_CG()‘ for clustering features and cells

Examples

```
data(celdaCGSim, celdaCGMod)
perplexity <- perplexity(celdaCGSim$counts, celdaCGMod)
```

perplexity,celda_G-method
Calculate the perplexity on new data with a celda_G model

Description

Perplexity is a statistical measure of how well a probability model can predict new data. Lower perplexity indicates a better model.

Usage

```
## S4 method for signature 'celda_G'
perplexity(counts, celdaMod, newCounts = NULL)
```

Arguments

counts	Integer matrix. Rows represent features and columns represent cells. This matrix should be the same as the one used to generate ‘celdaMod‘.
celdaMod	Celda object of class "celda_C"
newCounts	A new counts matrix used to calculate perplexity. If NULL, perplexity will be calculated for the ’counts’ matrix. Default NULL.

Value

Numeric. The perplexity for the provided count data and model.

See Also

‘celda_G()‘ for clustering features

Examples

```
data(celdaGSim, celdaGMod)
perplexity <- perplexity(celdaGSim$counts, celdaGMod)
```

---

plotDimReduceCluster *Plotting the cell labels on a dimensionality reduction plot*

Description

Create a scatterplot for each row of a normalized gene expression matrix where x and y axis are from a data dimensionality reduction tool. The cells are colored by its given ‘cluster‘ label.

Usage

```
plotDimReduceCluster(dim1, dim2, cluster, size = 1,
  xlab = "Dimension_1", ylab = "Dimension_2",
  specificClusters = NULL, labelClusters = FALSE, labelSize = 3.5)
```

Arguments

dim1	Numeric vector. First dimension from data dimensionality reduction output.
dim2	Numeric vector. Second dimension from data dimensionality reduction output.
cluster	Integer vector. Contains cluster labels for each cell.
size	Numeric. Sets size of point on plot. Default 1.
xlab	Character vector. Label for the x-axis. Default "Dimension_1".
ylab	Character vector. Label for the y-axis. Default "Dimension_2".
specificClusters	Numeric vector. Only color cells in the specified clusters. All other cells will be grey. If NULL, all clusters will be colored. Default NULL.
labelClusters	Logical. Whether the cluster labels are plotted. Default FALSE.
labelSize	Numeric. Sets size of label if labelClusters is TRUE. Default 3.5.

Value

The plot as a ggplot object

Examples

```
data(celdaCGSim, celdaCGMod)
celdaTsne <- celdaTsne(counts = celdaCGSim$counts,
    celdaMod = celdaCGMod)
plotDimReduceCluster(dim1 = celdaTsne[, 1],
    dim2 = celdaTsne[, 2],
    cluster = as.factor(clusters(celdaCGMod)$z),
    specificClusters = c(1, 2, 3))
```

---

plotDimReduceFeature *Plotting feature expression on a dimensionality reduction plot*

Description

Create a scatterplot for each row of a normalized gene expression matrix where x and y axis are from a data dimensionality reduction tool. The cells are colored by expression of the specified feature.

Usage

```
plotDimReduceFeature(dim1, dim2, counts, features, normalize = TRUE,
  exactMatch = TRUE, trim = c(-2, 2), size = 1,
  xlab = "Dimension_1", ylab = "Dimension_2", colorLow = "grey",
  colorMid = NULL, colorHigh = "blue")
```

Arguments

dim1	Numeric vector. First dimension from data dimensionality reduction output.
dim2	Numeric vector. Second dimension from data dimensionality reduction output.
counts	Integer matrix. Rows represent features and columns represent cells.
features	Character vector. Uses these genes for plotting.
normalize	Logical. Whether to normalize the columns of 'counts'. Default TRUE.
exactMatch	Logical. Whether an exact match or a partial match using 'grep()' is used to look up the feature in the rownames of the counts matrix. Default TRUE.
trim	Numeric vector. Vector of length two that specifies the lower and upper bounds for the data. This threshold is applied after row scaling. Set to NULL to disable. Default c(-2,2).
size	Numeric. Sets size of point on plot. Default 1.
xlab	Character vector. Label for the x-axis. Default "Dimension_1".
ylab	Character vector. Label for the y-axis. Default "Dimension_2".
colorLow	Character. A color available from 'colors()'. The color will be used to signify the lowest values on the scale. Default 'grey'.

colorMid	Character. A color available from 'colors()'. The color will be used to signify the midpoint on the scale.
colorHigh	Character. A color available from 'colors()'. The color will be used to signify the highest values on the scale. Default 'blue'.

Value

The plot as a ggplot object

Examples

```
data(celdaCGSim, celdaCGMod)
celdaTsne <- celdaTsne(counts = celdaCGSim$counts,
    celdaMod = celdaCGMod)
plotDimReduceFeature(dim1 = celdaTsne[, 1],
    dim2 = celdaTsne[, 2],
    counts = celdaCGSim$counts,
    features = c("Gene_99"),
    exactMatch = TRUE)
```

---

plotDimReduceGrid *Mapping the dimensionality reduction plot*

Description

Creates a scatterplot given two dimensions from a data dimensionality reduction tool (e.g tSNE) output.

Usage

```
plotDimReduceGrid(dim1, dim2, matrix, size, xlab, ylab, colorLow, colorMid,
  colorHigh, varLabel)
```

Arguments

dim1	Numeric vector. First dimension from data dimensionality reduction output.
dim2	Numeric vector. Second dimension from data dimensionality reduction output.
matrix	Numeric matrix. Each row of the matrix will be plotted as a separate facet.
size	Numeric. Sets size of point on plot. Default 1.
xlab	Character vector. Label for the x-axis. Default 'Dimension_1'.
ylab	Character vector. Label for the y-axis. Default 'Dimension_2'.
colorLow	Character. A color available from 'colors()'. The color will be used to signify the lowest values on the scale. Default 'grey'.
colorMid	Character. A color available from 'colors()'. The color will be used to signify the midpoint on the scale.
colorHigh	Character. A color available from 'colors()'. The color will be used to signify the highest values on the scale. Default 'blue'.
varLabel	Character vector. Title for the color legend.

Value

The plot as a ggplot object

Examples

```
data(celdaCGSim, celdaCGMod)
celdaTsne <- celdaTsne(counts = celdaCGSim$counts,
    celdaMod = celdaCGMod)
plotDimReduceGrid(celdaTsne[, 1],
    celdaTsne[, 2],
    matrix = celdaCGSim$counts,
    xlab = "Dimension1",
    ylab = "Dimension2",
    varLabel = "tsne",
    size = 1,
    colorLow = "grey",
    colorMid = NULL,
    colorHigh = "blue")
```

---

plotDimReduceModule *Plotting the Celda module probability on a dimensionality reduction plot*

Description

Create a scatterplot for each row of a normalized gene expression matrix where x and y axis are from a data dimensionality reduction tool. The cells are colored by the module probability(s).

Usage

```
plotDimReduceModule(dim1, dim2, counts, celdaMod, modules = NULL,
  rescale = TRUE, size = 1, xlab = "Dimension_1",
  ylab = "Dimension_2", colorLow = "grey", colorMid = NULL,
  colorHigh = "blue")
```

Arguments

dim1	Numeric vector. First dimension from data dimensionality reduction output.
dim2	Numeric vector. Second dimension from data dimensionality reduction output.
counts	Integer matrix. Rows represent features and columns represent cells. This matrix should be the same as the one used to generate 'celdaMod'.
celdaMod	Celda object of class "celda_G" or "celda_CG".
modules	Character vector. Module(s) from celda model to be plotted. e.g. c("1", "2").
rescale	Logical. Whether rows of the matrix should be rescaled to [0, 1]. Default TRUE.
size	Numeric. Sets size of point on plot. Default 1.
xlab	Character vector. Label for the x-axis. Default "Dimension_1".
ylab	Character vector. Label for the y-axis. Default "Dimension_2".
colorLow	Character. A color available from 'colors()'. The color will be used to signify the lowest values on the scale. Default 'grey'.

colorMid	Character. A color available from 'colors()'. The color will be used to signify the midpoint on the scale.
colorHigh	Character. A color available from 'colors()'. The color will be used to signify the highest values on the scale. Default 'blue'.

Value

The plot as a ggplot object

Examples

```
data(celdaCGSim, celdaCGMod)
celdaTsne <- celdaTsne(counts = celdaCGSim$counts,
    celdaMod = celdaCGMod)
plotDimReduceModule(
    dim1 = celdaTsne[, 1], dim2 = celdaTsne[, 2],
    counts = celdaCGSim$counts, celdaMod = celdaCGMod,
    modules = c("1", "2"))
```

---

plotGridSearchPerplexity

Visualize perplexity of a list of celda models

Description

Visualize perplexity of every model in a celdaList, by unique K/L combinations

Usage

```
plotGridSearchPerplexity(celdaList, sep = 1)
```

Arguments

celdaList	Object of class 'celdaList'.
sep	Numeric. Breaks in the x axis of the resulting plo.t.

Value

A ggplot plot object showing perplexity as a function of clustering parameters.

Examples

```
data(celdaCGSim, celdaCGGridSearchRes)
## Run various combinations of parameters with 'celdaGridSearch'
celdaCGGridSearchRes <- resamplePerplexity(
  celdaCGSim$counts,
  celdaCGGridSearchRes)
plotGridSearchPerplexity(celdaCGGridSearchRes)
```

plotGridSearchPerplexitycelda_C

Plot perplexity as a function of K from celda_C models

Description

Plots perplexity as a function of the cell (K) clusters as generated by celdaGridSearch().

Usage

```
plotGridSearchPerplexitycelda_C(celdaList, sep)
```

Arguments

celdaList	Object of class 'celdaList'.
sep	Numeric. Breaks in the x axis of the resulting plot.

Value

A ggplot plot object showing perplexity as a function of clustering parameters.

Examples

```
data(celdaCGSim, celdaCGGridSearchRes)
celdaCGGridSearchRes <- resamplePerplexity(
  celdaCGSim$counts,
  celdaCGGridSearchRes
)
plotGridSearchPerplexity(celdaCGGridSearchRes)
```

plotGridSearchPerplexitycelda_CG

Plot perplexity as a function of K and L from celda_CG models

Description

This function plots perplexity as a function of the cell/gene (K/L) clusters as generated by celdaGridSearch().

Usage

```
plotGridSearchPerplexitycelda_CG(celdaList, sep)
```

Arguments

celdaList	Object of class 'celdaList'.
sep	Numeric. Breaks in the x axis of the resulting plot.

Value

A ggplot plot object showing perplexity as a function of clustering parameters.

Examples

```
data(celdaCGSim, celdaCGGridSearchRes)
celdaCGGridSearchRes <- resamplePerplexity(
  celdaCGSim$counts,
  celdaCGGridSearchRes
)
plotGridSearchPerplexity(celdaCGGridSearchRes)
```

---

plotGridSearchPerplexitycelda_G

Plot perplexity as a function of L from a celda_G model

Description

Plots perplexity as a function of the gene (L) clusters as generated by celdaGridSearch().

Usage

```
plotGridSearchPerplexitycelda_G(celdaList, sep)
```

Arguments

celdaList	Object of class 'celdaList'.
sep	Numeric. Breaks in the x axis of the resulting plot.

Value

A ggplot plot object showing perplexity as a function of clustering parameters.

Examples

```
data(celdaCGSim, celdaCGGridSearchRes)
celdaCGGridSearchRes <- resamplePerplexity(
  celdaCGSim$counts,
  celdaCGGridSearchRes)
plotGridSearchPerplexity(celdaCGGridSearchRes)
```

---

plotHeatmap

Plots heatmap based on Celda model

Description

Renders a heatmap based on a matrix of counts where rows are features and columns are cells.

Usage

```
plotHeatmap(counts, z = NULL, y = NULL, scaleRow = scale,
  trim = c(-2, 2), featureIx = NULL, cellIx = NULL,
  clusterFeature = TRUE, clusterCell = TRUE,
  colorScheme = c("divergent", "sequential"),
  colorSchemeSymmetric = TRUE, colorSchemeCenter = 0, col = NULL,
  annotationCell = NULL, annotationFeature = NULL,
  annotationColor = NULL, breaks = NULL, legend = TRUE,
  annotationLegend = TRUE, annotationNamesFeature = TRUE,
  annotationNamesCell = TRUE, showNamesFeature = FALSE,
  showNamesCell = FALSE, hclustMethod = "ward.D2",
  treeheightFeature = ifelse(clusterFeature, 50, 0),
  treeheightCell = ifelse(clusterCell, 50, 0), silent = FALSE, ...)
```

Arguments

counts	Numeric matrix. Normalized counts matrix where rows represent features and columns represent cells. .
z	Numeric vector. Denotes cell population labels.
y	Numeric vector. Denotes feature module labels.
scaleRow	Function. A function to scale each individual row. Set to NULL to disable. Occurs after normalization and log transformation. Defualt is 'scale' and thus will Z-score transform each row.
trim	Numeric vector. Vector of length two that specifies the lower and upper bounds for the data. This threshold is applied after row scaling. Set to NULL to disable. Default c(-2,2).
featureIx	Integer vector. Select features for display in heatmap. If NULL, no subsetting will be performed. Default NULL.
cellIx	Integer vector. Select cells for display in heatmap. If NULL, no subsetting will be performed. Default NULL.
clusterFeature	Logical. Determines whether rows should be clustered. Default TRUE.
clusterCell	Logical. Determines whether columns should be clustered. Default TRUE.
colorScheme	"Character. One of ""divergent"" or ""sequential"". A ""divergent"" scheme is best for highlighting relative data (denoted by 'colorSchemeCenter') such as gene expression data that has been normalized and centered. A ""sequential"" scheme is best for highlighting data that are ordered low to high such as raw counts or probabilities. Default "divergent".
colorSchemeSymmetric	Logical. When the colorScheme is "divergent" and the data contains both positive and negative numbers, TRUE indicates that the color scheme should be symmetric from [-max(abs(data)), max(abs(data))]. For example, if the data ranges goes from -1.5 to 2, then setting this to TRUE will force the color scheme to range from -2 to 2. Default TRUE.
colorSchemeCenter	Numeric. Indicates the center of a "divergent" colorScheme. Default 0.
col	Color for the heatmap.
annotationCell	Data frame. Additional annotations for each cell will be shown in the column color bars. The format of the data frame should be one row for each cell and one column for each annotation. Numeric variables will be displayed as continuous color bars and factors will be displayed as discrete color bars. Default NULL.

annotationFeature
: A data frame for the feature annotations (rows).

annotationColor
: List. Contains color scheme for all annotations. See '?pheatmap' for more details.

breaks
: Numeric vector. A sequence of numbers that covers the range of values in the normalized 'counts'. Values in the normalized 'matrix' are assigned to each bin in 'breaks'. Each break is assigned to a unique color from 'col'. If NULL, then breaks are calculated automatically. Default NULL.

legend
: Logical. Determines whether legend should be drawn. Default TRUE.

annotationLegend
: Logical. Whether legend for all annotations should be drawn. Default TRUE.

annotationNamesFeature
: Logical. Whether the names for features should be shown. Default TRUE.

annotationNamesCell
: Logical. Whether the names for cells should be shown. Default TRUE.

showNamesFeature
: Logical. Specifies if feature names should be shown. Default TRUE.

showNamesCell
: Logical. Specifies if cell names should be shown. Default FALSE.

hclustMethod
: Character. Specifies the method to use for the 'hclust' function. See '?hclust' for possible values. Default "ward.D2".

treeheightFeature
: Numeric. Width of the feature dendrogram. Set to 0 to disable plotting of this dendrogram. Default: if clusterFeature == TRUE, then treeheightFeature = 50, else treeheightFeature = 0.

treeheightCell
: Numeric. Height of the cell dendrogram. Set to 0 to disable plotting of this dendrogram. Default: if clusterCell == TRUE, then treeheightCell = 50, else treeheightCell = 0.

silent
: Logical. Whether to plot the heatmap.

...
: Other arguments to be passed to underlying pheatmap function.

Value

list A list containing dendrogram information and the heatmap grob

Examples

```
data(celdaCGSim, celdaCGMod)
plotHeatmap(celdaCGSim$counts,
    z = clusters(celdaCGMod)$z, y = clusters(celdaCGMod)$y)
```

---

recodeClusterY *Recode feature module clusters*

Description

Recode feature module clusters using a mapping in the 'from' and 'to' arguments.

Usage

```
recodeClusterY(celdaMod, from, to)
```

Arguments

celdaMod	Celda object of class 'celda_G' or 'celda_CG'.
from	Numeric vector. Unique values in the range of seq(L) that correspond to the original cluster labels in 'celdaMod'.
to	Numeric vector. Unique values in the range of seq(L) that correspond to the new cluster labels.

Value

Celda object with recoded feature module clusters, with class corresponding to that of 'celdaMod'.

Examples

```
data(celdaCGMod)
celdaModReorderedY <- recodeClusterY(celdaCGMod, c(1, 3), c(3, 1))
```

---

recodeClusterZ *Recode cell cluster labels*

Description

Recode cell subpopulaton clusters using a mapping in the 'from' and 'to' arguments.

Usage

```
recodeClusterZ(celdaMod, from, to)
```

Arguments

celdaMod	Celda object of class 'celda_C' or 'celda_CG'.
from	Numeric vector. Unique values in the range of seq(K) that correspond to the original cluster labels in 'celdaMod'.
to	Numeric vector. Unique values in the range of seq(K) that correspond to the new cluster labels.

Value

Celda object with cell subpopulation clusters, with class corresponding to that of 'celdaMod'.

Examples

```
data(celdaCGMod)
celdaModReorderedZ <- recodeClusterZ(celdaCGMod, c(1, 3), c(3, 1))
```

`recursiveSplitCell` *Recursive cell splitting*

Description

Uses the 'celda_C' model to cluster cells into population for range of possible K's. The cell population labels of the previous "K-1" model are used as the initial values in the current model with K cell populations. The best split of an existing cell population is found to create the K-th cluster. This procedure is much faster than randomly initializing each model with a different K. If module labels for each feature are given in 'yInit', the 'celda_CG' model will be used to split cell populations based on those modules instead of individual features. Module labels will also be updated during sampling and thus may end up slightly different than 'yInit'.

Usage

```
recursiveSplitCell(counts, sampleLabel = NULL, initialK = 5,
  maxK = 25, tempL = NULL, yInit = NULL, alpha = 1, beta = 1,
  delta = 1, gamma = 1, minCell = 3, reorder = TRUE,
  perplexity = TRUE, logfile = NULL, verbose = TRUE)
```

Arguments

`counts`	Integer matrix. Rows represent features and columns represent cells.
`sampleLabel`	Vector or factor. Denotes the sample label for each cell (column) in the count matrix.
`initialK`	Integer. Minimum number of cell populations to try.
`maxK`	Integer. Maximum number of cell populations to try.
`tempL`	Integer. Number of temporary modules to identify and use in cell splitting. Only used if 'yInit = NULL'. Collapsing features to a relatively smaller number of modules will increase the speed of clustering and tend to produce better cell populations. This number should be larger than the number of true modules expected in the dataset. Default NULL.
`yInit`	Integer vector. Module labels for features. Cells will be clustered using the 'celda_CG' model based on the modules specified in 'yInit' rather than the counts of individual features. While the features will be initialized to the module labels in 'yInit', the labels will be allowed to move within each new model with a different K.
`alpha`	Numeric. Concentration parameter for Theta. Adds a pseudocount to each cell population in each sample. Default 1.
`beta`	Numeric. Concentration parameter for Phi. Adds a pseudocount to each feature in each cell (if 'yInit' is NULL) or to each module in each cell population (if 'yInit' is set). Default 1.
`delta`	Numeric. Concentration parameter for Psi. Adds a pseudocount to each feature in each module. Only used if 'yInit' is set. Default 1.
`gamma`	Numeric. Concentration parameter for Eta. Adds a pseudocount to the number of features in each module. Only used if 'yInit' is set. Default 1.
`minCell`	Integer. Only attempt to split cell populations with at least this many cells.

reorder	Logical. Whether to reorder cell populations using hierarchical clustering after each model has been created. If FALSE, cell populations numbers will correspond to the split which created the cell populations (i.e. 'K15' was created at split 15, 'K16' was created at split 16, etc.). Default TRUE.
perplexity	Logical. Whether to calculate perplexity for each model. If FALSE, then perplexity can be calculated later with 'resamplePerplexity()'. Default TRUE.
logfile	Character. Messages will be redirected to a file named 'logfile'. If NULL, messages will be printed to stdout. Default NULL.
verbose	Logical. Whether to print log messages. Default TRUE.

Value

Object of class 'celda_list', which contains results for all model parameter combinations and summaries of the run parameters. The models in the list will be of class 'celda_C' if 'yInit = NULL' or 'celda_CG' if 'zInit' is set.

See Also

'recursiveSplitModule()' for recursive splitting of cell populations.

Examples

```
data(celdaCGSim, celdaCSim)
## Create models that range from K = 3 to K = 7 by recursively splitting
## cell populations into two to produce `celda_C` cell clustering models
testZ <- recursiveSplitCell(celdaCSim$counts, initialK = 3, maxK = 7)

## Alternatively, first identify features modules using
## `recursiveSplitModule()`
moduleSplit <- recursiveSplitModule(celdaCGSim$counts,
    initialL = 3, maxL = 15)
plotGridSearchPerplexity(moduleSplit)
moduleSplitSelect <- subsetCeldaList(moduleSplit, list(L = 10))

## Then use module labels for initialization in `recursiveSplitCell()` to
## produce `celda_CG` bi-clustering models
cellSplit <- recursiveSplitCell(celdaCGSim$counts,
    initialK = 3, maxK = 7, yInit = clusters(moduleSplitSelect)$y)
plotGridSearchPerplexity(cellSplit)
celdaMod <- subsetCeldaList(cellSplit, list(K = 5, L = 10))
```

---

recursiveSplitModule *Recursive module splitting*

Description

Uses the 'celda_G' model to cluster features into modules for a range of possible L's. The module labels of the previous "L-1" model are used as the initial values in the current model with L modules. The best split of an existing module is found to create the L-th module. This procedure is much faster than randomly initializing each model with a different L.

Usage

```
recursiveSplitModule(counts, initialL = 10, maxL = 100, tempK = 100,
  zInit = NULL, sampleLabel = NULL, alpha = 1, beta = 1,
  delta = 1, gamma = 1, minFeature = 3, reorder = TRUE,
  perplexity = TRUE, verbose = TRUE, logfile = NULL)
```

Arguments

counts	Integer matrix. Rows represent features and columns represent cells.
initialL	Integer. Minimum number of modules to try.
maxL	Integer. Maximum number of modules to try.
tempK	Integer. Number of temporary cell populations to identify and use in module splitting. Only used if 'zInit=NULL' Collapsing cells to a relatively smaller number of cell popluations will increase the speed of module clustering and tend to produce better modules. This number should be larger than the number of true cell populations expected in the dataset. Default 100.
zInit	Integer vector. Collapse cells to cell populations based on labels in 'zInit' and then perform module splitting. If NULL, no collapasing will be performed unless 'tempK' is specified. Default NULL.
sampleLabel	Vector or factor. Denotes the sample label for each cell (column) in the count matrix. Only used if 'zInit' is set.
alpha	Numeric. Concentration parameter for Theta. Adds a pseudocount to each cell population in each sample. Only used if 'zInit' is set. Default 1.
beta	Numeric. Concentration parameter for Phi. Adds a pseudocount to each feature module in each cell. Default 1.
delta	Numeric. Concentration parameter for Psi. Adds a pseudocount to each feature in each module. Default 1.
gamma	Numeric. Concentration parameter for Eta. Adds a pseudocount to the number of features in each module. Default 1.
minFeature	Integer. Only attempt to split modules with at least this many features.
reorder	Logical. Whether to reorder modules using hierarchical clustering after each model has been created. If FALSE, module numbers will correspond to the split which created the module (i.e. 'L15' was created at split 15, 'L16' was created at split 16, etc.). Default TRUE.
perplexity	Logical. Whether to calculate perplexity for each model. If FALSE, then perplexity can be calculated later with 'resamplePerplexity()'. Default TRUE.
verbose	Logical. Whether to print log messages. Default TRUE.
logfile	Character. Messages will be redirected to a file named 'logfile'. If NULL, messages will be printed to stdout. Default NULL.

Value

Object of class 'celda_list', which contains results for all model parameter combinations and summaries of the run parameters. The models in the list will be of class 'celda_G' if 'zInit=NULL' or 'celda_CG' if 'zInit' is set.

See Also

'recursiveSplitCell()' for recursive splitting of cell populations.

Examples

```
data(celdaCGSim)
## Create models that range from L=3 to L=20 by recursively splitting modules
## into two
moduleSplit <- recursiveSplitModule(celdaCGSim$counts,
    initialL = 3, maxL = 20)

## Example results with perplexity
plotGridSearchPerplexity(moduleSplit)

## Select model for downstream analysis
celdaMod <- subsetCeldaList(moduleSplit, list(L = 10))
```

`resamplePerplexity` *Calculate and visualize perplexity of all models in a celdaList, with count resampling*

Description

Calculates the perplexity of each model's cluster assignments given the provided countMatrix, as well as resamplings of that count matrix, providing a distribution of perplexities and a better sense of the quality of a given K/L choice.

Usage

```
resamplePerplexity(counts, celdaList, resample = 5, seed = 12345)
```

Arguments

`counts`	Integer matrix. Rows represent features and columns represent cells. This matrix should be the same as the one used to generate 'celda.mod'.
`celdaList`	Object of class 'celdaList'.
`resample`	Integer. The number of times to resample the counts matrix for evaluating perplexity. Default 5.
`seed`	Integer. Passed to with_seed. For reproducibility, a default value of 12345 is used. If NULL, no calls to with_seed are made.

Value

celdaList. Returns the provided 'celdaList' with a 'perplexity' property, detailing the perplexity of all K/L combinations that appeared in the celdaList's models.

Examples

```
data(celdaCGSim, celdaCGGridSearchRes)
celdaCGGridSearchRes <- resamplePerplexity(
  celdaCGSim$counts,
  celdaCGGridSearchRes)
plotGridSearchPerplexity(celdaCGGridSearchRes)
```

resList *Get final celdaModels from a celdaList*

Description

Returns all models generated during a 'celdaGridSearch()' run.

Usage

```
resList(celdaList)
```

Arguments

celdaList	An object of class celdaList.

Value

List. Contains one celdaModel object for each of the parameters specified in the 'runParams()' of the provided celda list.

Examples

```
data(celdaCGGridSearchRes)
celdaCGGridModels <- resList(celdaCGGridSearchRes)
```

resList,celdaList-method *Get final celdaModels from a celdaList*

Description

Returns all models generated during a 'celdaGridSearch()' run.

Usage

```
## S4 method for signature 'celdaList'
resList(celdaList)
```

Arguments

celdaList	An object of class celdaList.

Value

List. Contains one celdaModel object for each of the parameters specified in the 'runParams()' of the provided celda list.

Examples

```
data(celdaCGGridSearchRes)
celdaCGGridModels <- resList(celdaCGGridSearchRes)
```

runParams *Get run parameters provided to 'celdaGridSearch()'*

Description

Returns details on the clustering parameters, and model priors provided to 'celdaGridSearch()' when the provided celdaList was created.

Usage

```
runParams(celdaList)
```

Arguments

celdaList	An object of class celdaList.

Value

Data Frame. Contains details on the various K/L parameters, chain parameters, and final log-likelihoods derived for each model in the provided celdaList.

Examples

```
data(celdaCGGridSearchRes)
runParams(celdaCGGridSearchRes)
```

runParams,celdaList-method *Get run parameters provided to 'celdaGridSearch()'*

Description

Returns details on the clustering parameters, and model priors provided to 'celdaGridSearch()' when the provided celdaList was created.

Usage

```
## S4 method for signature 'celdaList'
runParams(celdaList)
```

Arguments

celdaList	An object of class celdaList.

Value

Data Frame. Contains details on the various K/L parameters, chain parameters, and final log-likelihoods derived for each model in the provided celdaList.

Examples

```
data(celdaCGGridSearchRes)
runParams(celdaCGGridSearchRes)
```

sampleCells *sampleCells*

Description

A matrix of simulated gene counts.

Usage

```
sampleCells
```

Format

A matrix of simulated gene counts with 10 rows (genes) and 10 columns (cells).

Details

A toy count matrix for use with celda.

Generated by Josh Campbell.

Source

http://github.com/campbio/celda

sampleLabel *Get sampleLabels from a celdaModel*

Description

Returns the sampleLabels for the count matrix provided for generation of a given celdaModel.

Usage

```
sampleLabel(celdaMod)
```

Arguments

celdaMod	celdaModel. Options available in 'celda::availableModels'.

Value

Character. Contains the sampleLabels provided at model creation time, or those automatically generated by celda.

Examples

```
data(celdaCGMod)
sampleLabel(celdaCGMod)
```

sampleLabel,celdaModel-method *Get sampleLabels from a celdaModel*

Description

Returns the sampleLabels for the count matrix provided for generation of a given celdaModel.

Usage

```
## S4 method for signature 'celdaModel'
sampleLabel(celdaMod)
```

Arguments

celdaMod	celdaModel. Options available in 'celda::availableModels'.

Value

Character. Contains the sampleLabels provided at model creation time, or those automatically generated by celda.

Examples

```
data(celdaCGMod)
sampleLabel(celdaCGMod)
```

selectBestModel *Select best chain within each combination of parameters*

Description

Select the chain with the best log likelihood for each combination of tested parameters from a 'celdaList' object gererated by 'celdaGridSearch()'.

Usage

```
selectBestModel(celdaList, asList = FALSE)
```

Arguments

celdaList	Object of class 'celdaList'. An object containing celda models returned from 'celdaGridSearch()'.
asList	'TRUE' or 'FALSE'. Whether to return the best model as a 'celdaList' object or not. If 'FALSE', return the best model as a corresponding 'celda_C', 'celda_G' or 'celda_CG' object.

Value

A new ‘celdaList‘ object containing one model with the best log likelihood for each set of parameters. If only one set of parameters is in the ‘celdaList‘, the best model will be returned directly instead of a ‘celdaList‘ object.

See Also

‘celdaGridSearch()‘ can run Celda with multiple parameters and chains in parallel. ‘subsetCeldaList()‘ can subset the ‘celdaList‘ object.

Examples

```
data(celdaCGGridSearchRes)
## Returns same result as running celdaGridSearch with "bestOnly = TRUE"
cgsBest <- selectBestModel(celdaCGGridSearchRes)
```

---

semiPheatmap *A function to draw clustered heatmaps.*

Description

A function to draw clustered heatmaps where one has better control over some graphical parameters such as cell size, etc.

The function also allows to aggregate the rows using kmeans clustering. This is advisable if number of rows is so big that R cannot handle their hierarchical clustering anymore, roughly more than 1000. Instead of showing all the rows separately one can cluster the rows in advance and show only the cluster centers. The number of clusters can be tuned with parameter kmeansK.

Usage

```
semiPheatmap(mat, color = colorRampPalette(rev(brewer.pal(n = 7, name =
  "RdYlBu")))(100), kmeansK = NA, breaks = NA,
  borderColor = "grey60", cellWidth = NA, cellHeight = NA,
  scale = "none", clusterRows = TRUE, clusterCols = TRUE,
  clusteringDistanceRows = "euclidean",
  clusteringDistanceCols = "euclidean", clusteringMethod = "complete",
  clusteringCallback = .identity2, cutreeRows = NA, cutreeCols = NA,
  treeHeightRow = ifelse(clusterRows, 50, 0),
  treeHeightCol = ifelse(clusterCols, 50, 0), legend = TRUE,
  legendBreaks = NA, legendLabels = NA, annotationRow = NA,
  annotationCol = NA, annotation = NA, annotationColors = NA,
  annotationLegend = TRUE, annotationNamesRow = TRUE,
  annotationNamesCol = TRUE, dropLevels = TRUE, showRownames = TRUE,
  showColnames = TRUE, main = NA, fontSize = 10,
  fontSizeRow = fontSize, fontSizeCol = fontSize,
  displayNumbers = FALSE, numberFormat = "%.2f",
  numberColor = "grey30", fontSizeNumber = 0.8 * fontSize,
  gapsRow = NULL, gapsCol = NULL, labelsRow = NULL,
  labelsCol = NULL, fileName = NA, width = NA, height = NA,
  silent = FALSE, rowLabel, colLabel, ...)
```

Arguments

mat	numeric matrix of the values to be plotted.
color	vector of colors used in heatmap.
kmeansK	the number of kmeans clusters to make, if we want to agggregate the rows before drawing heatmap. If NA then the rows are not aggregated.
breaks	Numeric vector. A sequence of numbers that covers the range of values in the normalized ‘counts‘. Values in the normalized ‘matrix‘ are assigned to each bin in ‘breaks‘. Each break is assigned to a unique color from ‘col‘. If NULL, then breaks are calculated automatically. Default NULL.
borderColor	color of cell borders on heatmap, use NA if no border should be drawn.
cellWidth	individual cell width in points. If left as NA, then the values depend on the size of plotting window.
cellHeight	individual cell height in points. If left as NA, then the values depend on the size of plotting window.
scale	character indicating if the values should be centered and scaled in either the row direction or the column direction, or none. Corresponding values are "row", "column" and "none".
clusterRows	boolean values determining if rows should be clustered or hclust object,
clusterCols	boolean values determining if columns should be clustered or hclust object.
clusteringDistanceRows	distance measure used in clustering rows. Possible values are "correlation" for Pearson correlation and all the distances supported by dist, such as "euclidean", etc. If the value is none of the above it is assumed that a distance matrix is provided.
clusteringDistanceCols	distance measure used in clustering columns. Possible values the same as for clusteringDistanceRows.
clusteringMethod	clustering method used. Accepts the same values as hclust.
clusteringCallback	callback function to modify the clustering. Is called with two parameters: original hclust object and the matrix used for clustering. Must return a hclust object.
cutreeRows	number of clusters the rows are divided into, based on the hierarchical clustering (using cutree), if rows are not clustered, the argument is ignored
cutreeCols	similar to cutreeRows, but for columns
treeHeightRow	the height of a tree for rows, if these are clustered. Default value 50 points.
treeHeightCol	the height of a tree for columns, if these are clustered. Default value 50 points.
legend	logical to determine if legend should be drawn or not.
legendBreaks	vector of breakpoints for the legend.
legendLabels	vector of labels for the legendBreaks.
annotationRow	data frame that specifies the annotations shown on left side of the heatmap. Each row defines the features for a specific row. The rows in the data and in the annotation are matched using corresponding row names. Note that color schemes takes into account if variable is continuous or discrete.
annotationCol	similar to annotationRow, but for columns.

annotation	deprecated parameter that currently sets the annotationCol if it is missing.
annotationColors	list for specifying annotationRow and annotationCol track colors manually. It is possible to define the colors for only some of the features. Check examples for details.
annotationLegend	boolean value showing if the legend for annotation tracks should be drawn.
annotationNamesRow	boolean value showing if the names for row annotation tracks should be drawn.
annotationNamesCol	boolean value showing if the names for column annotation tracks should be drawn.
dropLevels	logical to determine if unused levels are also shown in the legend.
showRownames	boolean specifying if column names are be shown.
showColnames	boolean specifying if column names are be shown.
main	the title of the plot
fontSize	base fontsize for the plot
fontSizeRow	fontsize for rownames (Default: fontsize)
fontSizeCol	fontsize for colnames (Default: fontsize)
displayNumbers	logical determining if the numeric values are also printed to the cells. If this is a matrix (with same dimensions as original matrix), the contents of the matrix are shown instead of original values.
numberFormat	format strings (C printf style) of the numbers shown in cells. For example "%.2f" shows 2 decimal places and "%.1e" shows exponential notation (see more in sprintf).
numberColor	color of the text
fontSizeNumber	fontsize of the numbers displayed in cells
gapsRow	vector of row indices that show shere to put gaps into heatmap. Used only if the rows are not clustered. See cutreeRow to see how to introduce gaps to clustered rows.
gapsCol	similar to gapsRow, but for columns.
labelsRow	custom labels for rows that are used instead of rownames.
labelsCol	similar to labelsRow, but for columns.
fileName	file path where to save the picture. Filetype is decided by the extension in the path. Currently following formats are supported: png, pdf, tiff, bmp, jpeg. Even if the plot does not fit into the plotting window, the file size is calculated so that the plot would fit there, unless specified otherwise.
width	manual option for determining the output file width in inches.
height	manual option for determining the output file height in inches.
silent	do not draw the plot (useful when using the gtable output)
rowLabel	row cluster labels for semi-clustering
colLabel	column cluster labels for semi-clustering
...	graphical parameters for the text used in plot. Parameters passed to grid.text, see gpar.

Value

Invisibly a list of components

- `treeRow` the clustering of rows as `hclust` object
- `treeCol` the clustering of columns as `hclust` object
- `kmeans` the kmeans clustering of rows if parameter `kmeansK` was specified

Author(s)

Raivo Kolde <rkolde@gmail.com> #@examples # Create test matrix test = matrix(rnorm(200), 20, 10) test[seq(10), seq(1, 10, 2)] = test[seq(10), seq(1, 10, 2)] + 3 test[seq(11, 20), seq(2, 10, 2)] = test[seq(11, 20), seq(2, 10, 2)] + 2 test[seq(15, 20), seq(2, 10, 2)] = test[seq(15, 20), seq(2, 10, 2)] + 4 colnames(test) = paste("Test", seq(10), sep = "") rownames(test) = paste("Gene", seq(20), sep = "")

Draw heatmaps pheatmap(test) pheatmap(test, kmeansK = 2) pheatmap(test, scale = "row", clusteringDistanceRows = "correlation") pheatmap(test, color = colorRampPalette(c("navy", "white", "firebrick3"))(50)) pheatmap(test, cluster_row = FALSE) pheatmap(test, legend = FALSE)

Show text within cells pheatmap(test, displayNumbers = TRUE) pheatmap(test, displayNumbers = TRUE, numberFormat = "%.1e") pheatmap(test, displayNumbers = matrix(ifelse(test > 5, "*", ""), nrow(test))) pheatmap(test, cluster_row = FALSE, legendBreaks = seq(-1, 4), legendLabels = c("0", "1e-4", "1e-3", "1e-2", "1e-1", "1"))

Fix cell sizes and save to file with correct size pheatmap(test, cellWidth = 15, cellHeight = 12, main = "Example heatmap") pheatmap(test, cellWidth = 15, cellHeight = 12, fontSize = 8, fileName = "test.pdf")

Generate annotations for rows and columns annotationCol = data.frame(CellType = factor(rep(c("CT1", "CT2"), 5)), Time = seq(5)) rownames(annotationCol) = paste("Test", seq(10), sep = "")

annotationRow = data.frame(GeneClass = factor(rep(c("Path1", "Path2", "Path3"), c(10, 4, 6)))) rownames(annotationRow) = paste("Gene", seq(20), sep = "")

Display row and color annotations pheatmap(test, annotationCol = annotationCol) pheatmap(test, annotationCol = annotationCol, annotationLegend = FALSE) pheatmap(test, annotationCol = annotationCol, annotationRow = annotationRow)

Specify colors ann_colors = list(Time = c("white", "firebrick"), CellType = c(CT1 = "#1B9E77", CT2 = "#D95F02"), GeneClass = c(Path1 = "#7570B3", Path2 = "#E7298A", Path3 = "#66A61E"))

pheatmap(test, annotationCol = annotationCol, annotationColors = ann_colors, main = "Title") pheatmap(test, annotationCol = annotationCol, annotationRow = annotationRow, annotationColors = ann_colors) pheatmap(test, annotationCol = annotationCol, annotationColors = ann_colors[2])

Gaps in heatmaps pheatmap(test, annotationCol = annotationCol, clusterRows = FALSE, gapsRow = c(10, 14)) pheatmap(test, annotationCol = annotationCol, clusterRows = FALSE, gapsRow = c(10, 14), cutreeCol = 2)

Show custom strings as row/col names labelsRow = c("", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "Il10", "Il15", "Il1b")

pheatmap(test, annotationCol = annotationCol, labelsRow = labelsRow)

Specifying clustering from distance matrix drows = stats::dist(test, method = "minkowski") dcols = stats::dist(t(test), method = "minkowski") pheatmap(test, clusteringDistanceRows = drows, clusteringDistanceCols = dcols)

Modify ordering of the clusters using clustering callback option callback = function(hc, mat) sv = svd(t(mat))$v[, 1] dend = reorder(as.dendrogram(hc), wts = sv) as.hclust(dend)

pheatmap(test, clusteringCallback = callback)

simulateCells *Simulate count data from the celda generative models.*

Description

This function generates a list containing a simulated counts matrix, as well as various parameters used in the simulation which can be useful for running celda. The user must provide the desired model (one of celda_C, celda_G, celda_CG) as well as any desired tuning parameters for those model's simulation functions as detailed below.

Usage

```
simulateCells(model, ...)
```

Arguments

model	Character. Options available in 'celda::availableModels'.
...	Additional parameters.

Value

List. Contains the simulated counts matrix, derived cell cluster assignments, the provided parameters, and estimated Dirichlet distribution parameters for the model.

Examples

```
data(celdaCGSim)
dim(celdaCGSim$counts)
```

simulateCellscelda_C *Simulate cells from the celda_C model*

Description

Generates a simulated counts matrix, cell subpopulation clusters, and sample labels according to the generative process of the celda_C model.

Usage

```
simulateCellscelda_C(model, S = 5, CRange = c(50, 100),
  NRange = c(500, 1000), G = 100, K = 5, alpha = 1, beta = 1,
  seed = 12345, ...)
```

Arguments

model	Character. Options available in 'celda::availableModels'.
S	Integer. Number of samples to simulate. Default 5.
CRange	Vector of length 2 given the range (min, max) of number of cells for each sample to be randomly generated from the uniform distribution. Default c(50, 100).
NRange	Integer vector. A vector of length 2 that specifies the lower and upper bounds of the number of counts generated for each cell. Default c(500, 1000).
G	Integer. The total number of features to be simulated. Default 100.
K	Integer. Number of cell populations. Default 5.
alpha	Numeric. Concentration parameter for Theta. Adds a pseudocount to each cell population in each sample. Default 1.
beta	Numeric. Concentration parameter for Phi. Adds a pseudocount to each feature in each cell population. Default 1.
seed	Integer. Passed to with_seed. For reproducibility, a default value of 12345 is used. If NULL, no calls to with_seed are made.
...	Additional parameters.

Value

List. Contains the simulated matrix 'counts', cell population clusters 'z', sample assignments 'sampleLabel', and input parameters.

See Also

'celda_G()' for simulating feature modules and 'celda_CG()' for simulating feature modules and cell populations.

Examples

```
celdaCSim <- simulateCells(model = "celda_C", K = 10)
simCounts <- celdaCSim$counts
```

---

simulateCellscelda_CG *Simulate cells from the celda_CG model*

Description

Generates a simulated counts matrix, cell subpopulation clusters, sample labels, and feature module clusters according to the generative process of the celda_CG model.

Usage

```
simulateCellscelda_CG(model, S = 5, CRange = c(50, 100),
  NRange = c(500, 1000), G = 100, K = 5, L = 10, alpha = 1,
  beta = 1, gamma = 5, delta = 1, seed = 12345, ...)
```

Arguments

model	Character. Options available in 'celda::availableModels'.
S	Integer. Number of samples to simulate. Default 5.
CRange	Integer vector. A vector of length 2 that specifies the lower and upper bounds of the number of cells to be generated in each sample. Default c(50, 100).
NRange	Integer vector. A vector of length 2 that specifies the lower and upper bounds of the number of counts generated for each cell. Default c(500, 1000).
G	Integer. The total number of features to be simulated. Default 100.
K	Integer. Number of cell populations. Default 5.
L	Integer. Number of feature modules. Default 10.
alpha	Numeric. Concentration parameter for Theta. Adds a pseudocount to each cell population in each sample. Default 1.
beta	Numeric. Concentration parameter for Phi. Adds a pseudocount to each feature module in each cell population. Default 1.
gamma	Numeric. Concentration parameter for Eta. Adds a pseudocount to the number of features in each module. Default 5.
delta	Numeric. Concentration parameter for Psi. Adds a pseudocount to each feature in each module. Default 1.
seed	Integer. Passed to with_seed. For reproducibility, a default value of 12345 is used. If NULL, no calls to with_seed are made.
...	Additional parameters.

Value

List. Contains the simulated matrix 'counts', cell population clusters 'z', feature module clusters 'y', sample assignments 'sampleLabel', and input parameters.

See Also

'celda_C()' for simulating cell subpopulations and 'celda_G()' for simulating feature modules.

Examples

```
celdaCGSim <- simulateCells(model = "celda_CG")
```

---

simulateCellscelda_G *Simulate cells from the celda_G model*

Description

Generates a simulated counts matrix and feature module clusters according to the generative process of the celda_G model.

Usage

```
simulateCellscelda_G(model, C = 100, NRange = c(500, 1000), G = 100,
  L = 10, beta = 1, gamma = 5, delta = 1, seed = 12345, ...)
```

Arguments

model	Character. Options available in 'celda::availableModels'.
C	Integer. Number of cells to simulate. Default 100.
NRange	Integer vector. A vector of length 2 that specifies the lower and upper bounds of the number of counts generated for each cell. Default c(500, 5000).
G	Integer. The total number of features to be simulated. Default 100.
L	Integer. Number of feature modules. Default 10.
beta	Numeric. Concentration parameter for Phi. Adds a pseudocount to each feature module in each cell. Default 1.
gamma	Numeric. Concentration parameter for Eta. Adds a pseudocount to the number of features in each module. Default 5.
delta	Numeric. Concentration parameter for Psi. Adds a pseudocount to each feature in each module. Default 1.
seed	Integer. Passed to with_seed. For reproducibility, a default value of 12345 is used. If NULL, no calls to with_seed are made.
...	Additional parameters.

Value

List. Contains the simulated matrix 'counts', feature module clusters 'y', and input parameters.

See Also

'celda_C()' for simulating cell subpopulations and 'celda_CG()' for simulating feature modules and cell populations.

Examples

```
celdaGSim <- simulateCells(model = "celda_G")
```

---

simulateContaminatedMatrix

Simulate contaminated count matrix

Description

This function generates a list containing two count matrices – one for real expression, the other one for contamination, as well as other parameters used in the simulation which can be useful for running decontamination.

Usage

```
simulateContaminatedMatrix(C = 300, G = 100, K = 3, NRange = c(500,
  1000), beta = 0.5, delta = c(1, 2), seed = 12345)
```

Arguments

C	Integer. Number of cells to be simulated. Default to be 300.
G	Integer. Number of genes to be simulated. Default to be 100.
K	Integer. Number of cell populations to be simulated. Default to be 3.
NRange	Integer vector. A vector of length 2 that specifies the lower and upper bounds of the number of counts generated for each cell. Default to be c(500, 1000).
beta	Numeric. Concentration parameter for Phi. Default to be 0.5.
delta	Numeric or Numeric vector. Concentration parameter for Theta. If input as a single numeric value, symmetric values for beta distribution are specified; if input as a vector of lenght 2, the two values will be the shape1 and shape2 paramters of the beta distribution respectively.
seed	Integer. Passed to with_seed. For reproducibility, a default value of 12345 is used. If NULL, no calls to with_seed are made.

Value

A list object containing the real expression matrix and contamination expression matrix as well as other parameters used in the simulation.

Examples

```
contaminationSim <- simulateContaminatedMatrix(K = 3, delta = c(1, 9))
contaminationSim <- simulateContaminatedMatrix(K = 3, delta = 1)
```

---

subsetCeldaList *Subset celdaList object from celdaGridSearch*

Description

Select a subset of models from a ‘celdaList‘ object generated by ‘celdaGridSearch()‘ that match the criteria in the argument ‘params‘.

Usage

```
subsetCeldaList(celdaList, params)
```

Arguments

celdaList	celdaList Object of class ‘celdaList‘. An object containing celda models returned from ‘celdaGridSearch‘.
params	List. List of parameters used to subset celdaList.

Value

A new ‘celdaList‘ object containing all models matching the provided criteria in ‘params‘. If only one item in the ‘celdaList‘ matches the given criteria, the matching model will be returned directly instead of a ‘celdaList‘ object.

See Also

'celdaGridSearch()' can run Celda with multiple parameters and chains in parallel. 'selectBestModel()' can get the best model for each combination of parameters.

Examples

```
data(celdaCGGridSearchRes)
resK5L10 <- subsetCeldaList(celdaCGGridSearchRes, params = list(K = 5,
    L = 10))
```

---

topRank *Identify features with the highest influence on clustering.*

Description

topRank() can quickly identify the top 'n' rows for each column of a matrix. For example, this can be useful for identifying the top 'n' features per cell.

Usage

```
topRank(matrix, n = 25, margin = 2, threshold = 0,
  decreasing = TRUE)
```

Arguments

matrix	Numeric matrix.
n	Integer. Maximum number of items above 'threshold' returned for each ranked row or column.
margin	Integer. Dimension of 'matrix' to rank, with 1 for rows, 2 for columns. Default 2.
threshold	Numeric. Only return ranked rows or columns in the matrix that are above this threshold. If NULL, then no threshold will be applied. Default 1.
decreasing	Logical. Specifies if the rank should be decreasing. Default TRUE.

Value

List. The 'index' variable provides the top 'n' row (feature) indices contributing the most to each column (cell). The 'names' variable provides the rownames corresponding to these indexes.

Examples

```
data(sampleCells)
topRanksPerCell <- topRank(sampleCells, n = 5)
topFeatureNamesForCell <- topRanksPerCell$names[1]
```

violinPlot *Feature Expression Violin Plot*

Description

Outputs a violin plot for feature expression data.

Usage

```
violinPlot(counts, celdaMod, features, plotDots = FALSE)
```

Arguments

counts	Integer matrix. Rows represent features and columns represent cells.
celdaMod	Celda object of class "celda_G" or "celda_CG".
features	Character vector. Uses these genes for plotting.
plotDots	**TRUE** or **FALSE**. If **TRUE**, the expression of features will be plotted as points in addition to the violin curve.

Value

Violin plot for each feature, grouped by celda cluster

Examples

```
data(celdaCGSim, celdaCGMod)
violinPlot(counts = celdaCGSim$counts,
    celdaMod = celdaCGMod, features = "Gene_1")
```

Index