Package ‘scDblFinder’

May 18, 2024

Type Package

Title scDblFinder

Version 1.18.0

URL https://github.com/plger/scDblFinder,
https://plger.github.io/scDblFinder/

BugReports https://github.com/plger/scDblFinder/issues

Description The scDblFinder package gathers various methods for the detection and handling of doublets/multiplets in single-cell sequencing data (i.e. multiple cells captured within the same droplet or reaction volume). It includes methods formerly found in the scran package, the new fast and comprehensive scDblFinder method, and a reimplementation of the Amulet detection method for single-cell ATAC-seq.

License GPL-3 + file LICENSE

Depends R (>= 4.0), SingleCellExperiment

Imports igraph, Matrix, BiocGenerics, BiocParallel, BiocNeighbors, BiocSingular, S4Vectors, SummarizedExperiment, scran, scater, scuttle, bluster, methods, DelayedArray, xgboost, stats, utils, MASS, IRanges, GenomicRanges, GenomeInfoDb, Rsamtools, rtracklayer

Suggests BiocStyle, knitr, rmarkdown, testthat, scRNAseq, circlize, ComplexHeatmap, ggplot2, dplyr, viridisLite, mbkmeans

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.2.3

biocViews Preprocessing, SingleCell, RNASeq, ATACSeq

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

git_branch RELEASE_3_19

git_last_commit 226a100

git_last_commit_date 2024-04-30

Repository Bioconductor 3.19

Date/Publication 2024-05-17

Author Pierre-Luc Germain [cre, aut] (<https://orcid.org/0000-0003-3418-4218>),
Aaron Lun [ctb]

Maintainer Pierre-Luc Germain <pierre-luc.germain@hest.ethz.ch>

Contents

`addDoublets` *addDoublets*

Description

Adds artificial doublets to an existing dataset

Usage

```
addDoublets(
  x,
  clusters,
  dbr = (0.01 * ncol(x)/1000),
  only.heterotypic = TRUE,
  adjustSize = FALSE,
  prefix = "doublet.",
  ...
)
```

Arguments

`x`	A count matrix of singlets, or a `SummarizedExperiment-class`
`clusters`	A vector of cluster labels for each column of ‘x‘
`dbr`	The doublet rate
`only.heterotypic`	Whether to add only heterotypic doublets.
`adjustSize`	Whether to adjust the library sizes of the doublets.
`prefix`	Prefix for the colnames generated.
`...`	Any further arguments to `createDoublets`.

Value

A ‘SingleCellExperiment‘ with the colData columns ‘cluster‘ and ‘type‘ (indicating whether the cell is a singlet or doublet).

Examples

```
sce <- mockDoubletSCE(dbl.rate=0)
sce <- addDoublets(sce, clusters=sce$cluster)
```

`aggregateFeatures` *aggregateFeatures*

Description

Aggregates similar features (rows).

Usage

```
aggregateFeatures(
  x,
  dims.use = seq(2L, 12L),
  k = 1000,
  num_init = 3,
  use.mbk = NULL,
  use.subset = 20000,
  minCount = 1L,
  norm.fn = TFIDF,
  twoPass = FALSE,
  ...
)
```

Arguments

`x`	A integer/numeric (sparse) matrix, or a ‘SingleCellExperiment‘ including a ‘counts‘ assay.
`dims.use`	The PCA dimensions to use for clustering rows.
`k`	The approximate number of meta-features desired
`num_init`	The number of initializations used for k-means clustering.
`use.mbk`	Logical; whether to use minibatch k-means (see `mbkmeans`). If NULL, the minibatch approach will be used if there are more than 30000 features.
`use.subset`	How many cells (columns) to use to cluster the features.
`minCount`	The minimum number of counts for a region to be included.
`norm.fn`	The normalization function to use on the un-clustered data (a function taking a count matrix as a single argument and returning a matrix of the same dimensions). TFIDF by default.
`twoPass`	Logical; whether to perform the procedure twice, so in the second round cells are aggregated based on the meta-features of the first round, before re-clustering the features. Ignored if the dataset has fewer than ‘use.subset‘ cells.
`...`	Passed to `mbkmeans`. Can for instance be used to pass the ‘BPPARAM‘ argument for multithreading.

Value

An aggregated version of ‘x‘ (either an array or a ‘SingleCellExperiment‘, depending on the input). If ‘x‘ is a ‘SingleCellExperiment‘, the feature clusters will also be stored in ‘metadata(x)$featureGroups‘

`amulet` *amulet*

Description

ATACseq (Thibodeau, Eroglu, et al., Genome Biology 2021). The rationale is that cells with unexpectedly many loci covered by more than two reads are more likely to be doublets.

Usage

```
amulet(x, ...)
```

Arguments

`x`	The path to a fragments file, or a GRanges object containing the fragments (with the ‘name‘ column containing the barcode, and the ‘score‘ column containing the count).
`...`	Any argument to `getFragmentOverlaps`.

Details

When used on normal (or compressed) fragment files, this implementation is relatively fast (except for reading in the data) but it has a large memory footprint since the overlaps are performed in memory. It is therefore recommended to compress the fragment files using bgzip and index them with Tabix; in this case each chromosome will be read and processed separately, leading to a considerably lower memory footprint. See the underlying `getFragmentOverlaps` for details.

Value

A data.frame including, for each barcode, the number sites covered by more than two reads, the number of reads, and p- and q-values (low values indicative of doublets).

Examples

```
# here we use a dummy fragment file for example:
fragfile <- system.file( "extdata", "example_fragments.tsv.gz",
                         package="scDblFinder" )
res <- amulet(fragfile)
```

`amuletFromCounts` *amuletFromCounts*

Description

A reimplementation of the Amulet doublet detection method for single-cell ATACseq (Thibodeau, Eroglu, et al., Genome Biology 2021), based on tile/peak counts. Note that this is only a fast approximation to the original Amulet method, and *performs considerably worse*; for an equivalent implementation, see `amulet`.

Usage

```
amuletFromCounts(x, maxWidth = 500L, exclude = c("chrM", "M", "Mt"))
```

Arguments

`x`	A ‘SingleCellExperiment‘ object, or a matrix of counts with cells as columns. If the rows represent peaks, it is recommended to limite their width (see details).
`maxWidth`	the maximum width for a feature to be included. This is ignored unless ‘x‘ is a ‘SingleCellExperiment‘ with ‘rowRanges‘.
`exclude`	an optional ‘GRanges‘ of regions to be excluded. This is ignored unless ‘x‘ is a ‘SingleCellExperiment‘ with ‘rowRanges‘.

Details

The rationale for the amulet method is that a single diploid cell should not have more than two reads covering a single genomic location, and the method looks for cells enriched with sites covered by more than two reads. If the method is applied on a peak-level count matrix, however, larger peaks can however contain multiple reads even though no single nucleotide is covered more than once. Therefore, in such case we recommend to limit the width of the peaks used for this analysis, ideally to maximum twice the upper bound of the fragment size. For example, with a mean fragment size of 250bp and standard deviation of 125bp, peaks larger than 500bp are very likely to contain non-overlapping fragments, and should therefore be excluded using the ‘maxWidth‘ argument.

Value

If ‘x‘ is a ‘SingleCellExperiment‘, returns the object with an additional ‘amuletFromCounts.q‘ colData column. Otherwise returns a vector of the amulet doublet q-values for each cell.

See Also

`amulet`

Examples

```
x <- mockDoubletSCE()
x <- amuletFromCounts(x)
table(call=x$amuletFromCounts.q<0.05, truth=x$type)
```

clamulet *clamulet*

Description

Classification-powered Amulet-like method

Usage

```
clamulet(
  x,
  artificialDoublets = NULL,
  iter = 2,
  k = NULL,
  minCount = 0.001,
  maxN = 500,
  nfeatures = 25,
  max_depth = 5,
  threshold = 0.75,
  returnAll = FALSE,
  verbose = TRUE,
  ...
)
```

Arguments

`x`	The path to a fragment file (see `getFragmentOverlaps` for performance/memory-related guidelines)
`artificialDoublets`	The number of artificial doublets to generate
`iter`	The number of learning iterations (should be 1 to)
`k`	The number(s) of nearest neighbors at which to gather statistics
`minCount`	The minimum number of cells in which a locus is detected to be considered. If lower than 1, it is interpreted as a fraction of the number of cells.
`maxN`	The maximum number of regions per cell to consider to establish windows for meta-features
`nfeatures`	The number of meta-features to consider
`max_depth`	The maximum tree depth
`threshold`	The score threshold used during iterations
`returnAll`	Logical; whether to return data also for artificial doublets
`verbose`	Logical; whether to print progress information
`...`	Arguments passed to `getFragmentOverlaps`

Details

'clamulet' operates similarly to the 'scDblFinder' method, but generates doublets by operating on the fragment coverages. This has the advantage that the number of loci covered by more than two reads can be computed for artificial doublets, enabling the use of this feature (along with the kNN-based ones) in a classification scheme. It however has the disadvantage of being rather slow and memory hungry, and appears to be outperformed by a simple p-value combination of the two methods (see vignette).

Value

A data.frame

`clusterStickiness` *clusterStickiness*

Description

Tests for enrichment of doublets created from each cluster (i.e. cluster's stickiness). Only applicable with >=4 clusters. Note that when applied to an multisample object, this functions assumes that the cluster labels match across samples.

Usage

```
clusterStickiness(
  x,
  type = c("quasibinomial", "nbinom", "binomial", "poisson"),
  inclDiff = NULL,
  verbose = TRUE
)
```

Arguments

`x`	A table of double statistics, or a SingleCellExperiment on which scDblFinder was run using the cluster-based approach.
`type`	The type of test to use (quasibinomial recommended).
`inclDiff`	Logical; whether to include the difficulty in the model. If NULL, will be used only if there is a significant trend with the enrichment.
`verbose`	Logical; whether to print additional running information.

Value

A table of test results for each cluster.

Examples

```
sce <- mockDoubletSCE(rep(200,5))
sce <- scDblFinder(sce, clusters=TRUE, artificialDoublets=500)
clusterStickiness(sce)
```

`computeDoubletDensity` *Compute the density of simulated doublets*

Description

Identify potential doublet cells based on the local density of simulated doublet expression profiles. This replaces the older `doubletCells` function from the **scran** package.

Usage

```
computeDoubletDensity(x, ...)

## S4 method for signature 'ANY'
computeDoubletDensity(
  x,
  size.factors.norm = NULL,
  size.factors.content = NULL,
  k = 50,
  subset.row = NULL,
  niters = max(10000, ncol(x)),
  block = 10000,
  dims = 25,
  BNPARAM = KmknnParam(),
  BSPARAM = bsparam(),
  BPPARAM = SerialParam()
)

## S4 method for signature 'SummarizedExperiment'
computeDoubletDensity(x, ..., assay.type = "counts")

## S4 method for signature 'SingleCellExperiment'
computeDoubletDensity(x, size.factors.norm = sizeFactors(x), ...)
```

Arguments

`x`	A numeric matrix-like object of count values, where each column corresponds to a cell and each row corresponds to an endogenous gene. Alternatively, a SummarizedExperiment or SingleCellExperiment object containing such a matrix.
`...`	For the generic, additional arguments to pass to specific methods. For the SummarizedExperiment and SingleCellExperiment methods, additional arguments to pass to the ANY method.
`size.factors.norm`	A numeric vector of size factors for normalization of `x` prior to PCA and distance calculations. If `NULL`, defaults to size factors derived from the library sizes of `x`. For the SingleCellExperiment method, the default values are taken from `sizeFactors(x)`, if they are available.

`size.factors.content`	A numeric vector of size factors for RNA content normalization of `x` prior to simulating doublets. This is orthogonal to the values in `size.factors.norm`, see Details.
`k`	An integer scalar specifying the number of nearest neighbours to use to determine the bandwidth for density calculations.
`subset.row`	See ?"`scran-gene-selection`".
`niters`	An integer scalar specifying how many simulated doublets should be generated.
`block`	An integer scalar controlling the rate of doublet generation, to keep memory usage low.
`dims`	An integer scalar specifying the number of components to retain after the PCA.
`BNPARAM`	A BiocNeighborParam object specifying the nearest neighbor algorithm. This should be an algorithm supported by `findNeighbors`.
`BSPARAM`	A BiocSingularParam object specifying the algorithm to use for PCA, if `d` is not `NA`.
`BPPARAM`	A BiocParallelParam object specifying whether the neighbour searches should be parallelized.
`assay.type`	A string specifying which assay values contain the count matrix.

Details

This function simulates doublets by adding the count vectors for two randomly chosen cells in `x`. For each original cell, we compute the density of neighboring simulated doublets and compare it to the density of neighboring original cells. Genuine doublets should have a high density of simulated doublets relative to the density of its neighbourhood. Thus, the doublet score for each cell is defined as the ratio of densities of simulated doublets to the density of the original cells.

Densities are calculated in low-dimensional space after a PCA on the log-normalized expression matrix of `x`. Simulated doublets are projected into the low-dimensional space using the rotation vectors computed from the original cells. For each cell, the density of simulated doublets is computed for a hypersphere with radius set to the median distance to the `k` nearest neighbour. This is normalized by `niters`, `k` and the total number of cells in `x` to yield the final score.

The two size factor arguments have different roles:

- `size.factors.norm` contains the size factors to be used for normalization prior to PCA and distance calculations. This defaults to the values returned by `librarySizeFactors` but can be explicitly set to ensure that the low-dimensional space is consistent with that in the rest of the analysis.
- `size.factors.content` is much more important, and represents the size factors that preserve RNA content differences. This is usually computed from spike-in RNA and ensures that the simulated doublets have the correct ratio of contributions from the original cells.

It is possible to set both of these arguments as they are orthogonal to each other. Setting `size.factors.content` will not affect the calculation of log-normalized expression values from `x`. Conversely, setting `size.factors.norm` will not affect the ratio in which cells are added together when simulating doublets.

Value

A numeric vector of doublet scores for each cell in x.

Author(s)

Aaron Lun

References

Lun ATL (2018). Detecting doublet cells with *scran*. https://ltla.github.io/SingleCellThoughts/software/doublet_detection/bycell.html

See Also

findDoubletClusters, to detect doublet clusters.

scDblFinder, which uses a hybrid approach involving simulation and overclustering.

More detail on the mathematical background of this function is provided in the corresponding vignette at `vignette("computeDoubletDensity", package="scDblFinder")`.

Examples

```
# Mocking up an example.
set.seed(100)
ngenes <- 1000
mu1 <- 2^rnorm(ngenes)
mu2 <- 2^rnorm(ngenes)
mu3 <- 2^rnorm(ngenes)
mu4 <- 2^rnorm(ngenes)

counts.1 <- matrix(rpois(ngenes*100, mu1), nrow=ngenes) # Pure type 1
counts.2 <- matrix(rpois(ngenes*100, mu2), nrow=ngenes) # Pure type 2
counts.3 <- matrix(rpois(ngenes*100, mu3), nrow=ngenes) # Pure type 3
counts.4 <- matrix(rpois(ngenes*100, mu4), nrow=ngenes) # Pure type 4
counts.m <- matrix(rpois(ngenes*20, mu1+mu2), nrow=ngenes) # Doublets (1 & 2)

counts <- cbind(counts.1, counts.2, counts.3, counts.4, counts.m)
clusters <- rep(1:5, c(rep(100, 4), ncol(counts.m)))

# Find potential doublets.
scores <- computeDoubletDensity(counts)
boxplot(split(log10(scores), clusters))
```

createDoublets *createDoublets*

Description

Creates artificial doublet cells by combining given pairs of cells

Usage

```
createDoublets(
  x,
  dbl.idx,
  clusters = NULL,
  resamp = 0.5,
  halfSize = 0.5,
  adjustSize = FALSE,
  prefix = "dbl."
)
```

Arguments

`x`	A count matrix of real cells
`dbl.idx`	A matrix or data.frame with pairs of cell indexes stored in the first two columns.
`clusters`	An optional vector of cluster labels (for each column of ‘x‘)
`resamp`	Logical; whether to resample the doublets using the poisson distribution. Alternatively, if a proportion between 0 and 1, the proportion of doublets to resample.
`halfSize`	Logical; whether to half the library size of doublets (instead of just summing up the cells). Alternatively, a number between 0 and 1 can be given, determining the proportion of the doublets for which to perform the size adjustment. Ignored if not resampling.
`adjustSize`	Logical; whether to adjust the size of the doublets using the median sizes per cluster of the originating cells. Requires ‘clusters‘ to be given. Alternatively to a logical value, a number between 0 and 1 can be given, determining the proportion of the doublets for which to perform the size adjustment.
`prefix`	Prefix for the colnames generated.

Value

A matrix of artificial doublets.

Examples

```
sce <- mockDoubletSCE()
idx <- getCellPairs(sce$cluster, n=200)
art.dbls <- createDoublets(sce, idx)
```

cxds2 *cxds2*

Description

Calculates a coexpression-based doublet score using the method developed by Bais and Kostka 2020. This is the original implementation from the scds package, but enabling scores to be calculated for all cells while the gene coexpression is based only on a subset (i.e. excluding known/artificial doublets) and making it robust to low sparsity.

Usage

```
cxds2(x, whichDbls = c(), ntop = 500, binThresh = NULL)
```

Arguments

`x`	A matrix of counts, or a ‘SingleCellExperiment‘ containing a ’counts’
`whichDbls`	The columns of ‘x‘ which are known doublets.
`ntop`	The number of top features to keep.
`binThresh`	The count threshold to be considered expressed.

Value

A cxds score or, if ‘x‘ is a ‘SingleCellExperiment‘, ‘x‘ with an added ‘cxds_score‘ colData column.

References

`https://doi.org/10.1093/bioinformatics/btz698`

Examples

```
sce <- mockDoubletSCE()
sce <- cxds2(sce)
# which is equivalent to
# sce$cxds_score <- cxds2(counts(sce))
```

directDblClassification *directClassification*

Description

Trains a classifier directly on the expression matrix to distinguish artificial doublets from real cells.

Usage

```
directDblClassification(
  sce,
  dbr = NULL,
  processing = "default",
  iter = 2,
  dims = 20,
  nrounds = 0.25,
  max_depth = 6,
  ...
)
```

Arguments

`sce`	A `SummarizedExperiment-class`, `SingleCellExperiment-class`, or array of counts.
`dbr`	The expected doublet rate. By default this is assumed to be 1% per thousand cells captured (so 4% among 4000 thousand cells), which is appropriate for 10x datasets. Corrections for homeotypic doublets will be performed on the given rate.
`processing`	Counts (real and artificial) processing. Either 'default' (normal `scater`-based normalization and PCA), "rawPCA" (PCA without normalization), "rawFeatures" (no normalization/dimensional reduction), "normFeatures" (uses normalized features, without PCA) or a custom function with (at least) arguments 'e' (the matrix of counts) and 'dims' (the desired number of dimensions), returning a named matrix with cells as rows and components as columns.
`iter`	A positive integer indicating the number of scoring iterations. At each iteration, real cells that would be called as doublets are excluding from the training, and new scores are calculated.
`dims`	The number of dimensions used.
`nrounds`	Maximum rounds of boosting. If NULL, will be determined through cross-validation.
`max_depth`	Maximum depths of each tree.
`...`	Any doublet generation or pre-processing argument passed to 'scDblFinder'.

Value

A SummarizedExperiment-class with the additional ‘colData‘ column ‘directDoubletScore‘.

Examples

```
sce <- directDblClassification(mockDoubletSCE(), artificialDoublets=1)
boxplot(sce$directDoubletScore~sce$type)
```

---

doubletPairwiseEnrichment

doubletPairwiseEnrichment

Description

Calculates enrichment in any type of doublet (i.e. specific combination of clusters) over random expectation. Note that when applied to an multisample object, this functions assumes that the cluster labels match across samples.

Usage

```
doubletPairwiseEnrichment(
  x,
  lower.tail = FALSE,
  sampleWise = FALSE,
  type = c("poisson", "binomial", "nbinom", "chisq"),
  inclDiff = TRUE,
  verbose = TRUE
)
```

Arguments

`x`	A table of double statistics, or a SingleCellExperiment on which scDblFinder was run using the cluster-based approach.
`lower.tail`	Logical; defaults to FALSE to test enrichment (instead of depletion).
`sampleWise`	Logical; whether to perform tests sample-wise in multi-sample datasets. If FALSE (default), will aggregate counts before testing.
`type`	Type of test to use.
`inclDiff`	Logical; whether to regress out any effect of the identification difficulty in calculating expected counts
`verbose`	Logical; whether to output eventual warnings/notes

Value

A table of significances for each combination.

Examples

```
sce <- mockDoubletSCE()
sce <- scDblFinder(sce, clusters=TRUE, artificialDoublets=500)
doubletPairwiseEnrichment(sce)
```

---

doubletThresholding *doubletThresholding*

Description

Sets the doublet scores threshold; typically called by scDblFinder.

Usage

```
doubletThresholding(
  d,
  dbr = NULL,
  dbr.sd = NULL,
  stringency = 0.5,
  p = 0.1,
  method = c("auto", "optim", "dbr", "griffiths"),
  perSample = TRUE,
  returnType = c("threshold", "call")
)
```

Arguments

`d`	A data.frame of cell properties, with each row representing a cell, as produced by ‘scDblFinder(..., returnType="table")‘, or minimally containing a ‘score‘ column.
`dbr`	The expected (mean) doublet rate. If ‘d‘ contains a ‘cluster‘ column, the doublet rate will be adjusted for homotypic doublets.
`dbr.sd`	The standard deviation of the doublet rate, representing the uncertainty in the estimate. Ignored if ‘method!="optim"‘.
`stringency`	A numeric value >0 and <1 which controls the relative weight of false positives (i.e. real cells) and false negatives (artificial doublets) in setting the threshold. A value of 0.5 gives equal weight to both; a higher value (e.g. 0.7) gives higher weight to the false positives, and a lower to artificial doublets. Ignored if ‘method!="optim"‘.
`p`	The p-value threshold determining the deviation in doublet score.
`method`	The thresholding method to use, either ’auto’ (default, automatic selection depending on the available fields), ’optim’ (optimization of misclassification rate and deviation from expected doublet rate), ’dbr’ (strictly based on the expected doublet rate), or ’griffiths’ (cluster-wise number of median absolute deviation in doublet score).

`perSample`	Logical; whether to perform thresholding individually for each sample.
`returnType`	The type of value to return, either doublet calls (‘call‘) or thresholds (‘threshold‘).

Value

A vector of doublet calls if ‘returnType=="call"‘, or a threshold (or vector of thresholds) if ‘returnType=="threshold"‘.

Examples

```
sce <- mockDoubletSCE()
d <- scDblFinder(sce, verbose=FALSE, returnType="table")
th <- doubletThresholding(d, dbr=0.05)
th
```

`fastcluster` *fastcluster*

Description

Performs a fast two-step clustering: first clusters using k-means with a very large k, then uses louvain clustering of the k cluster averages and reports back the cluster labels.

Usage

```
fastcluster(
  x,
  k = NULL,
  rdname = "PCA",
  nstart = 3,
  iter.max = 50,
  ndims = NULL,
  nfeatures = 1000,
  verbose = TRUE,
  returnType = c("clusters", "preclusters", "metacells", "graph"),
  ...
)
```

Arguments

`x`	An object of class SCE
`k`	The number of k-means clusters to use in the primary step (should be much higher than the number of expected clusters). Defaults to 1/10th of the number of cells with a maximum of 3000.
`rdname`	The name of the dimensionality reduction to use.

`nstart`	Number of starts for k-means clustering
`iter.max`	Number of iterations for k-means clustering
`ndims`	Number of dimensions to use
`nfeatures`	Number of features to use (ignored if ‘rdname‘ is given and the corresponding dimensional reduction exists in ‘sce‘)
`verbose`	Logical; whether to output progress messages
`returnType`	See return.
`...`	Arguments passed to ‘scater::runPCA‘ (e.g. BPPARAM or BSPARAM) if ‘x‘ does not have ‘rdname‘.

Value

By default, a vector of cluster labels. If ‘returnType='preclusters'‘, returns the k-means pre-clusters. If ‘returnType='metacells'‘, returns the metacells aggretated by pre-clusters and the corresponding cell indexes. If ‘returnType='graph'‘, returns the graph of (meta-)cells and the corresponding cell indexes.

Examples

```
sce <- mockDoubletSCE()
sce$cluster <- fastcluster(sce)
```

`findDoubletClusters` *Detect doublet clusters*

Description

Identify potential clusters of doublet cells based on whether they have intermediate expression profiles, i.e., their profiles lie between two other “source” clusters.

Usage

```
findDoubletClusters(x, ...)

## S4 method for signature 'ANY'
findDoubletClusters(
  x,
  clusters,
  subset.row = NULL,
  threshold = 0.05,
  get.all.pairs = FALSE,
  ...
)

## S4 method for signature 'SummarizedExperiment'
```

```
findDoubletClusters(x, ..., assay.type = "counts")

## S4 method for signature 'SingleCellExperiment'
findDoubletClusters(x, clusters = colLabels(x, onAbsence = "error"), ...)
```

Arguments

`x`	A numeric matrix-like object of count values, where each column corresponds to a cell and each row corresponds to an endogenous gene. Alternatively, a SummarizedExperiment or SingleCellExperiment object containing such a matrix.
`...`	For the generic, additional arguments to pass to specific methods. For the ANY method, additional arguments to pass to `findMarkers`. For the SummarizedExperiment method, additional arguments to pass to the ANY method. For the SingleCellExperiment method, additional arguments to pass to the SummarizedExperiment method.
`clusters`	A vector of length equal to `ncol(x)`, containing cluster identities for all cells. If `x` is a SingleCellExperiment, this is taken from `colLabels(x)` by default.
`subset.row`	See `?"scran-gene-selection"`.
`threshold`	A numeric scalar specifying the FDR threshold with which to identify significant genes.
`get.all.pairs`	Logical scalar indicating whether statistics for all possible source pairings should be returned.
`assay.type`	A string specifying which assay values to use, e.g., `"counts"` or `"logcounts"`.

Details

This function detects clusters of doublet cells in a manner similar to the method used by Bach et al. (2017). For each "query" cluster, we examine all possible pairs of "source" clusters, hypothesizing that the query consists of doublets formed from the two sources. If so, gene expression in the query cluster should be strictly intermediate between the two sources after library size normalization.

We apply pairwise t-tests to the normalized log-expression profiles to reject this null hypothesis. This is done by identifying genes that are consistently up- or down-regulated in the query compared to *both* sources. We count the number of genes that reject the null hypothesis at the specified FDR `threshold`. For each query cluster, the most likely pair of source clusters is that which minimizes the number of significant genes.

Potential doublet clusters are identified using the following characteristics, in order of importance:

- Low number of significant genes (i.e., `num.de`). Ideally, `median.de` is also high to indicate that the absence of strong DE is not due to a lack of power.
- A reasonable proportion of cells in the cluster, i.e., `prop`. This requires some expectation of the doublet rate in the experimental protocol.
- Library sizes of the source clusters that are below that of the query cluster, i.e., `lib.size*` values below unity. This assumes that the doublet cluster will contain more RNA and have more counts than either of the two source clusters.

For each query cluster, the function will only report the pair of source clusters with the lowest `num.de`. Setting `get.all.pairs=TRUE` will retrieve statistics for all pairs of potential source clusters. This can be helpful for diagnostics to identify relationships between specific clusters.

The reported `p.value` is of little use in a statistical sense, and is only provided for inspection. Technically, it could be treated as the Simes combined p-value against the doublet hypothesis for the query cluster. However, this does not account for the multiple testing across all pairs of clusters for each chosen cluster, especially as we are chosing the pair that is most concordant with the doublet null hypothesis.

We use library size normalization (via `librarySizeFactors`) even if existing size factors are present. This is because intermediate expression of the doublet cluster is not guaranteed for arbitrary size factors. For example, expression in the doublet cluster will be higher than that in the source clusters if normalization was performed with spike-in size factors.

Value

A DataFrame containing one row per query cluster with the following fields:

`source1`: String specifying the identity of the first source cluster.

`source2`: String specifying the identity of the second source cluster.

`num.de`: Integer, number of genes that are significantly non-intermediate in the query cluster compared to the two putative source clusters.

`median.de`: Integer, median number of genes that are significantly non-intermediate in the query cluster across all possible source cluster pairings.

`best`: String specifying the identify of the top gene with the lowest p-value against the doublet hypothesis for this combination of query and source clusters.

`p.value`: Numeric, containing the adjusted p-value for the `best` gene.

`lib.size1`: Numeric, ratio of the median library sizes for the first source cluster to the query cluster.

`lib.size2`: Numeric, ratio of the median library sizes for the second source cluster to the query cluster.

`prop`: Numeric, proportion of cells in the query cluster.

`all.pairs`: A SimpleList object containing the above statistics for every pair of potential source clusters, if `get.all.pairs=TRUE`.

Each row is named according to its query cluster.

Author(s)

Aaron Lun

References

Bach K, Pensa S, Grzelak M, Hadfield J, Adams DJ, Marioni JC and Khaled WT (2017). Differentiation dynamics of mammary epithelial cells revealed by single-cell RNA sequencing. *Nat Commun.* 8, 1:2128.

See Also

findMarkers, to detect DE genes between clusters.

Examples

```
# Mocking up an example.
library(SingleCellExperiment)
sce <- mockDoubletSCE(c(200,300,200))

# Compute doublet-ness of each cluster:
dbl <- findDoubletClusters(counts(sce), sce$cluster)
dbl

# Narrow this down to clusters with very low 'N':
library(scuttle)
isOutlier(dbl$num.de, log=TRUE, type="lower")

# Get help from "lib.size" below 1.
dbl$lib.size1 < 1 & dbl$lib.size2 < 1
```

---

getArtificialDoublets *getArtificialDoublets*

Description

Create expression profiles of random artificial doublets.

Usage

```
getArtificialDoublets(
  x,
  n = 3000,
  clusters = NULL,
  resamp = 0.25,
  halfSize = 0.25,
  adjustSize = 0.25,
  propRandom = 0.1,
  selMode = c("proportional", "uniform", "sqrt"),
  n.meta.cells = 2,
  meta.triplets = TRUE,
  trim.q = c(0.05, 0.95)
)
```

Arguments

`x`	A count matrix, with features as rows and cells as columns.
`n`	The approximate number of doublet to generate (default 3000).
`clusters`	The optional clusters labels to use to build cross-cluster doublets.
`resamp`	Logical; whether to resample the doublets using the poisson distribution. Alternatively, if a proportion between 0 and 1, the proportion of doublets to resample.
`halfSize`	Logical; whether to half the library size of doublets (instead of just summing up the cells). Alternatively, a number between 0 and 1 can be given, determining the proportion of the doublets for which to perform the size adjustment.
`adjustSize`	Logical; whether to adjust the size of the doublets using the ratio between each cluster's median library size. Alternatively, a number between 0 and 1 can be given, determining the proportion of the doublets for which to perform the size adjustment.
`propRandom`	The proportion of the created doublets that are fully random (default 0.1); the rest will be doublets created across clusters. Ignored if ‘clusters‘ is NULL.
`selMode`	The cell pair selection mode for inter-cluster doublet generation, either 'uniform' (same number of doublets for each combination), 'proportional' (proportion expected from the clusters' prevalences), or 'sqrt' (roughly the square root of the expected proportion).
`n.meta.cells`	The number of meta-cell per cluster to create. If given, additional doublets will be created from cluster meta-cells. Ignored if ‘clusters‘ is missing.
`meta.triplets`	Logical; whether to create triplets from meta cells. Ignored if ‘clusters‘ is missing.
`trim.q`	A vector of two values between 0 and 1

Value

A list with two elements: ‘counts‘ (the count matrix of the artificial doublets) and ‘origins‘ the clusters from which each artificial doublets originated (NULL if ‘clusters‘ is not given).

Examples

```
m <- t(sapply( seq(from=0, to=5, length.out=50),
               FUN=function(x) rpois(30,x) ) )
doublets <- getArtificialDoublets(m, 30)
```

`getCellPairs` *getCellPairs*

Description

Given a vector of cluster labels, returns pairs of cross-cluster cells

Usage

```
getCellPairs(
  clusters,
  n = 1000,
  ls = NULL,
  q = c(0.1, 0.9),
  selMode = "proportional",
  soft.min = 5
)
```

Arguments

`clusters`	A vector of cluster labels for each cell, or a list containing metacells and graph
`n`	The number of cell pairs to obtain
`ls`	Optional library sizes
`q`	Library size quantiles between which to include cells (ignored if ‘ls‘ is NULL)
`selMode`	How to decide the number of pairs of each kind to produce. Either ’proportional’ (default, proportional to the abundance of the underlying clusters), ’uniform’ or ’sqrt’.
`soft.min`	Minimum number of pairs of a given type.

Value

A data.frame with the columns

Examples

```
# create random labels
x <- sample(head(LETTERS), 100, replace=TRUE)
getCellPairs(x, n=6)
```

`getExpectedDoublets` *getExpectedDoublets*

Description

getExpectedDoublets

Usage

```
getExpectedDoublets(x, dbr = NULL, only.heterotypic = TRUE)
```

Arguments

`x`	A vector of cluster labels for each cell
`dbr`	The expected doublet rate.
`only.heterotypic`	Logical; whether to return expectations only for heterotypic doublets

Value

The expected number of doublets of each combination of clusters

Examples

```
# random cluster labels
cl <- sample(head(LETTERS,4), size=2000, prob=c(.4,.2,.2,.2), replace=TRUE)
getExpectedDoublets(cl)
```

getFragmentOverlaps   *getFragmentOverlaps*

Description

Count the number of overlapping fragments.

Usage

```
getFragmentOverlaps(
  x,
  barcodes = NULL,
 regionsToExclude = GRanges(c("M", "chrM", "MT", "X", "Y", "chrX", "chrY"), IRanges(1L,
    width = 10^8)),
  minFrags = 500L,
  uniqueFrags = TRUE,
  maxFragSize = 1000L,
  removeHighOverlapSites = TRUE,
  fullInMemory = FALSE,
  BPPARAM = NULL,
  verbose = TRUE,
  ret = c("stats", "loci", "coverages")
)
```

Arguments

`x`	The path to a fragments file, or a GRanges object containing the fragments (with the ‘name‘ column containing the barcode, and optionally the ‘score‘ column containing the count).
`barcodes`	Optional character vector of cell barcodes to consider

`regionsToExclude`	A GRanges of regions to exclude. As per the original Amulet method, we recommend excluding repeats, as well as sex and mitochondrial chromosomes.
`minFrags`	Minimum number of fragments for a barcode to be considered. If ‘uniqueFrags=TRUE‘, this is the minimum number of unique fragments. Ignored if ‘barcodes‘ is given.
`uniqueFrags`	Logical; whether to use only unique fragments.
`maxFragSize`	Integer indicating the maximum fragment size to consider
`removeHighOverlapSites`	Logical; whether to remove sites that have more than two reads in unexpectedly many cells.
`fullInMemory`	Logical; whether to process all chromosomes together. This will speed up the process but at the cost of a very high memory consumption (as all fragments will be loaded in memory). This is anyway the default mode when ‘x‘ is not Tabix-indexed.
`BPPARAM`	A ‘BiocParallel‘ parameter object for multithreading. Note that multithreading will increase the memory usage.
`verbose`	Logical; whether to print progress messages.
`ret`	What to return, either barcode ’stats’ (default), ’loci’, or ’coverages’.

Details

When used on normal (or compressed) fragment files, this implementation is relatively fast (except for reading in the data) but it has a large memory footprint since the overlaps are performed in memory. It is therefore recommended to compress the fragment files using bgzip and index them with Tabix; in this case each chromosome will be read and processed separately, leading to a considerably lower memory footprint.

Value

A data.frame with counts and overlap statistics for each barcode.

`mockDoubletSCE` *mockDoubletSCE*

Description

Creates a mock random single-cell experiment object with doublets

Usage

```
mockDoubletSCE(
  ncells = c(200, 300),
  ngenes = 200,
  mus = NULL,
  dbl.rate = 0.1,
  only.heterotypic = TRUE
)
```

Arguments

`ncells`	A positive integer vector indicating the number of cells per cluster (min 2 clusters)
`ngenes`	The number of genes to simulate. Ignored if ‘mus‘ is given.
`mus`	A list of cluster averages.
`dbl.rate`	The doublet rate
`only.heterotypic`	Whether to create only heterotypic doublets

Value

A SingleCellExperiment object, with the colData columns ‘type‘ indicating whether the cell is a singlet or doublet, and ‘cluster‘ indicating from which cluster (or cluster combination) it was simulated.

Examples

```
sce <- mockDoubletSCE()
```

`plotDoubletMap` *plotDoubletMap*

Description

Plots a heatmap of observed versus expected doublets. Requires the ‘ComplexHeatmap‘ package.

Usage

```
plotDoubletMap(
  sce,
  colorBy = "enrichment",
  labelBy = "observed",
  addSizes = TRUE,
  col = NULL,
  column_title = "Clusters",
  row_title = "Clusters",
  column_title_side = "bottom",
  na_col = "white",
  ...
)
```

Arguments

`sce`	A SingleCellExperiment object on which ‘scDblFinder‘ has been run with the cluster-based approach.
`colorBy`	Determines the color mapping. Either "enrichment" (for log2-enrichment over expectation) or any column of ‘metadata(sce)$scDblFinder.stats‘
`labelBy`	Determines the cell labels. Either "enrichment" (for log2-enrichment over expectation) or any column of ‘metadata(sce)$scDblFinder.stats‘
`addSizes`	Logical; whether to add the sizes of clusters to labels
`col`	The colors scale to use (passed to ‘ComplexHeatmap::Heatmap‘)
`column_title`	passed to ‘ComplexHeatmap::Heatmap‘
`row_title`	passed to ‘ComplexHeatmap::Heatmap‘
`column_title_side`	passed to ‘ComplexHeatmap::Heatmap‘
`na_col`	color for NA cells
`...`	passed to ‘ComplexHeatmap::Heatmap‘

Value

a Heatmap object

`plotThresholds` *plotThresholds*

Description

Plots scores used for thresholding.

Usage

```
plotThresholds(d, ths = (0:100)/100, dbr = NULL, dbr.sd = NULL, do.plot = TRUE)
```

Arguments

`d`	A data.frame of cell properties, with each row representing a cell, as produced by ‘scDblFinder(..., returnType="table")‘.
`ths`	A vector of thresholds between 0 and 1 at which to plot values.
`dbr`	The expected (mean) doublet rate.
`dbr.sd`	The standard deviation of the doublet rate, representing the uncertainty in the estimate.
`do.plot`	Logical; whether to plot the data (otherwise will return the underlying data.frame).

Value

A ggplot, or a data.frame if ‘do.plot==FALSE‘.

`propHomotypic` *propHomotypic*

Description

Computes the proportion of pairs expected to be made of elements from the same cluster.

Usage

```
propHomotypic(clusters)
```

Arguments

`clusters`	A vector of cluster labels

Value

A numeric value between 0 and 1.

Examples

```
clusters <- sample(LETTERS[1:5], 100, replace=TRUE)
propHomotypic(clusters)
```

`recoverDoublets` *Recover intra-sample doublets*

Description

Recover intra-sample doublets that are neighbors to known inter-sample doublets in a multiplexed experiment.

Usage

```
recoverDoublets(x, ...)

## S4 method for signature 'ANY'
recoverDoublets(
  x,
  doublets,
  samples,
  k = 50,
  transposed = FALSE,
  subset.row = NULL,
  BNPARAM = KmknnParam(),
  BPPARAM = SerialParam()
)

## S4 method for signature 'SummarizedExperiment'
recoverDoublets(x, ..., assay.type = "logcounts")

## S4 method for signature 'SingleCellExperiment'
recoverDoublets(x, ..., use.dimred = NULL)
```

Arguments

`x`	A log-expression matrix for all cells (including doublets) in columns and genes in rows. If `transposed=TRUE`, this should be a matrix of low-dimensional coordinates where each row corresponds to a cell. Alternatively, a SummarizedExperiment or SingleCellExperiment containing (i) a log-expression matrix in the `assays` as specified by `assay.type`, or (ii) a matrix of reduced dimensions in the `reducedDims` as specified by `use.dimred`.
`...`	For the generic, additional arguments to pass to specific methods. For the SummarizedExperiment method, additional arguments to pass to the ANY method. For the SingleCellExperiment method, additional arguments to pass to the SummarizedExperiment method.
`doublets`	A logical, integer or character vector specifying which cells in `x` are known (inter-sample) doublets.
`samples`	A numeric vector containing the relative proportions of cells from each sample, used to determine how many cells are to be considered as intra-sample doublets.
`k`	Integer scalar specifying the number of nearest neighbors to use for computing the local doublet proportions.
`transposed`	Logical scalar indicating whether `x` is transposed, i.e., cells in the rows.
`subset.row`	A logical, integer or character vector specifying the genes to use for the neighbor search. Only used when `transposed=FALSE`.
`BNPARAM`	A BiocNeighborParam object specifying the algorithm to use for the nearest neighbor search.
`BPPARAM`	A BiocParallelParam object specifying the parallelization to use for the nearest neighbor search.
`assay.type`	A string specifying which assay values contain the log-expression matrix.
`use.dimred`	A string specifying whether existing values in `reducedDims(x)` should be used.

Details

In multiplexed single-cell experiments, we can detect doublets as libraries with labels for multiple samples. However, this approach fails to identify doublets consisting of two cells with the same label. Such cells may be problematic if they are still sufficiently abundant to drive formation of spurious clusters.

This function identifies intra-sample doublets based on the similarity in expression profiles to known inter-sample doublets. For each cell, we compute the proportion of the `k` neighbors that are known doublets. Of the “unmarked” cells that are not known doublets, those with top X largest proportions are considered to be intra-sample doublets. We use `samples` to obtain a reasonable estimate for X, see the vignette for details.

A larger value of `k` provides more stable estimates of the doublet proportion in each cell. However, this comes at the cost of assuming that each cell actually has `k` neighboring cells of the same state. For example, if a doublet cluster has fewer than `k` members, its doublet proportions will be “diluted” by inclusion of unmarked cells in the next-closest cluster.

Value

A DataFrame containing one row per cell and the following fields:

- `proportion`, a numeric field containing the proportion of neighbors that are doublets.
- `known`, a logical field indicating whether this cell is a known inter-sample doublet.
- `predicted`, a logical field indicating whether this cell is a predicted intra-sample doublet.

The `metadata` contains `intra`, a numeric scalar containing the expected number of intra-sample doublets.

Author(s)

Aaron Lun

See Also

`doubletCells` and `doubletCluster`, for alternative methods of doublet detection when no prior doublet information is available.

`hashedDrops` from the **DropletUtils** package, to identify doublets from cell hashing experiments.

More detail on the mathematical background of this function is provided in the corresponding vignette at `vignette("recoverDoublets", package="scDblFinder")`.

Examples

```
# Mocking up an example.
set.seed(100)
ngenes <- 1000
mu1 <- 2^rnorm(ngenes, sd=2)
mu2 <- 2^rnorm(ngenes, sd=2)

counts.1 <- matrix(rpois(ngenes*100, mu1), nrow=ngenes) # Pure type 1
counts.2 <- matrix(rpois(ngenes*100, mu2), nrow=ngenes) # Pure type 2
counts.m <- matrix(rpois(ngenes*20, mu1+mu2), nrow=ngenes) # Doublets (1 & 2)
all.counts <- cbind(counts.1, counts.2, counts.m)
lcounts <- scuttle::normalizeCounts(all.counts)

# Pretending that half of the doublets are known. Also pretending that
# the experiment involved two samples of equal size.
known <- 200 + seq_len(10)
out <- recoverDoublets(lcounts, doublets=known, k=10, samples=c(1, 1))
out
```

scDblFinder *scDblFinder*

Description

Identification of heterotypic (or neotypic) doublets in single-cell RNAseq using cluster-based generation of artificial doublets.

Usage

```
scDblFinder(
  sce,
  clusters = NULL,
  samples = NULL,
  clustCor = NULL,
  artificialDoublets = NULL,
  knownDoublets = NULL,
  knownUse = c("discard", "positive"),
  dbr = NULL,
  dbr.sd = NULL,
  nfeatures = 1352,
  dims = 20,
  k = NULL,
  removeUnidentifiable = TRUE,
  includePCs = 19,
  propRandom = 0,
  propMarkers = 0,
  aggregateFeatures = FALSE,
  returnType = c("sce", "table", "full", "counts"),
  score = c("xgb", "weighted", "ratio"),
  processing = "default",
  metric = "logloss",
  nrounds = 0.25,
  max_depth = 4,
  iter = 3,
  trainingFeatures = NULL,
  unident.th = NULL,
 multiSampleMode = c("split", "singleModel", "singleModelSplitThres", "asOne"),
  threshold = TRUE,
  verbose = is.null(samples),
  BPPARAM = SerialParam(),
  ...
)
```

Arguments

sce	A SummarizedExperiment-class, SingleCellExperiment-class, or array of counts.

`clusters`	The optional cluster assignments. This is used to make doublets more efficiently. `clusters` should either be a vector of labels for each cell, or the name of a colData column of `sce`. Alternatively, if 'clusters=TRUE', fast clustering will be performed. If 'clusters' is a single integer, it will determine how many clusters to create (using k-means clustering). If 'clusters' is NULL or FALSE, purely random artificial doublets will be generated.
`samples`	A vector of the same length as cells (or the name of a column of `colData(x)`), indicating to which sample each cell belongs. Here, a sample is understood as being processed independently. If omitted, doublets will be searched for with all cells together. If given, doublets will be searched for independently for each sample, which is preferable if they represent different captures. If your samples were multiplexed using cell hashes, what you want to give here are the different batches/wells (i.e. independent captures, since doublets cannot arise across them) rather than biological samples.
`clustCor`	Include Spearman correlations to cell type averages in the predictors. If 'clustCor' is a matrix of cell type marker expressions (with features as rows and cell types as columns), the subset of these which are present in the selected features will be correlated to each cell to produce additional predictors (i.e. one per cell type). Alternatively, if 'clustCor' is a positive integer, this number of inter-cluster markers will be selected and used for correlation (se 'clustCor=Inf' to use all available genes).
`artificialDoublets`	The approximate number of artificial doublets to create. If `NULL`, will be the maximum of the number of cells or `5*nbClusters^2` (with a minimum of 1500).
`knownDoublets`	An optional logical vector of known doublets (e.g. through cell barcodes), or the name of a colData column of 'sce' containing that information. The way these are used depends on the 'knownUse' argument.
`knownUse`	The way to use known doublets, either 'discard' (they are discarded for the purpose of training, but counted as positive for thresholding) or 'positive' (they are used as positive doublets for training - usually leads to a mild decrease in accuracy due to the fact that known doublets typically include a sizeable fraction of homotypic doublets). Note that 'scDblFinder' does *not* enforce that the knownDoublets be necessarily called as doublets in the final classification, if they are not predicted as such.
`dbr`	The expected doublet rate. By default this is assumed to be 1% per thousand cells captured (so 4% among 4000 thousand cells), which is appropriate for 10x datasets. Corrections for homeotypic doublets will be performed on the given rate.
`dbr.sd`	The uncertainty range in the doublet rate, interpreted as a +/- around 'dbr'. During thresholding, deviation from the expected doublet rate will be calculated from these boundaries, and will be considered null within these boundaries. If NULL, will be 40% of 'dbr'. Set to 'dbr.sd=0' to disable the uncertainty around the doublet rate, or to 'dbr.sd=1' to disable any expectation of the number of doublets (thus letting the thresholding be entirely driven by the misclassification of artificial doublets).
`nfeatures`	The number of top features to use. Alternatively, a character vectors of feature names (e.g. highly-variable genes) to use.

`dims`	The number of dimensions used.
`k`	Number of nearest neighbors (for KNN graph). If more than one value is given, the doublet density will be calculated at each k (and other values at the highest k), and all the information will be used by the classifier. If omitted, a reasonable set of values is used.
`removeUnidentifiable`	Logical; whether to remove artificial doublets of a combination that is generally found to be unidentifiable.
`includePCs`	The index of principal components to include in the predictors (e.g. ‘includePCs=1:2‘), or the number of top components to use (e.g. ‘includePCs=10‘, equivalent to 1:10).
`propRandom`	The proportion of the artificial doublets which should be made of random cells (as opposed to inter-cluster combinations). If clusters is FALSE or NULL, this is ignored (and set to 1).
`propMarkers`	The proportion of features to select based on marker identification.
`aggregateFeatures`	Whether to perform feature aggregation (recommended for ATAC). Can also be a positive integer, in which case this will indicate the number of components to use for feature aggregation (if TRUE, ‘dims‘ will be used.)
`returnType`	Either "sce" (default), "table" (to return the table of cell attributes including artificial doublets), or "full" (returns an SCE object containing both the real and artificial cells).
`score`	Score to use for final classification.
`processing`	Counts (real and artificial) processing before KNN. Either ’default’ (normal `scater`-based normalization and PCA), "rawPCA" (PCA without normalization), "rawFeatures" (no normalization/dimensional reduction), "normFeatures" (uses normalized features, without PCA) or a custom function with (at least) arguments ‘e‘ (the matrix of counts) and ‘dims‘ (the desired number of dimensions), returning a named matrix with cells as rows and components as columns.
`metric`	Error metric to optimize during training (e.g. ’merror’, ’logloss’, ’auc’, ’aucpr’).
`nrounds`	Maximum rounds of boosting. If NULL, will be determined through cross-validation. If a number <=1, will used the best cross-validation round minus ‘nrounds‘ times the standard deviation of the classification error.
`max_depth`	Maximum depths of each tree.
`iter`	A positive integer indicating the number of scoring iterations (ignored if ‘score‘ isn’t based on classifiers). At each iteration, real cells that would be called as doublets are excluding from the training, and new scores are calculated. Recommended values are 1 or 2.
`trainingFeatures`	The features to use for training (defaults to an optimal pre-selection based on benchmark datasets). To exclude features (rather than list those to be included), prefix them with a "-".
`unident.th`	The score threshold below which artificial doublets will be considered unidentifiable.

`multiSampleMode`	Either "split" (recommended if there is heterogeneity across samples), "singleModel", "singleModelSplitThres", or "asOne" (see details below).
`threshold`	Logical; whether to threshold scores into binary doublet calls
`verbose`	Logical; whether to print messages and the thresholding plot.
`BPPARAM`	Used for multithreading when splitting by samples (i.e. when ‘samples!=NULL‘); otherwise passed to eventual PCA and K/SNN calculations.
`...`	further arguments passed to `getArtificialDoublets`.

Details

This function generates artificial doublets from real cells, evaluates their prevalence in the neighborhood of each cells, and uses this along with additional cell-level features to classify doublets. The approach is complementary to doublets identified via cell hashes and SNPs in multiplexed samples: the latter can identify doublets formed by cells of the same type from two samples, which are nearly undistinguishable from real cells transcriptionally, but cannot identify doublets made by cells of the same sample. See `vignette("scDblFinder")` for more details on the method.

The ‘clusters‘ and ‘propRandom‘ argument determines whether the artificial doublets are generated between clusters or randomly.

When multiple samples/captures are present, they should be specified using the `samples` argument. In this case, we recommend the use of `BPPARAM` to perform several of the steps in parallel. Artificial doublets and kNN networks will be computed separately; then the behavior will then depend on the ‘multiSampleMode‘ argument:

- *split*: the whole process is split by sample. This is the default and recommended mode, because it is the most robust (e.g. to heterogeneity between samples, also for instance in the number of cells), and in practice we have not seen major gains in sharing information across samples;
- *singleModel*: the doublets are generated on a per-sample basis, but the classifier and thresholding will be trained globally;
- *singleModelSplitThres*: the doublets are generated on a per-sample basis, the classifier is trained globally, but the final thresholding is per-sample;
- *asOne*: the doublet rate (if not given) is calculated as the weighted average of sample-specific doublet rates, and all samples are otherwise run as if they were one sample. This can get computationally more intensive, and can lead to biases if there are batch effects.

When inter-sample doublets are available, they can be provided to ‘scDblFinder‘ through the `knownDoublets` argument to improve the identification of further doublets. How exactly these are used depends on the ‘knownUse‘ argument: with ’discard’ (default), the known doublets are excluded from the training step, but counted as positives. With ’positive’, they are included and treated as positive doublets for the training step. Note that because known doublets can in practice include a lot of homotypic doublets, this second approach can often lead to a slight decrease in the accuracy of detecting heterotypic doublets.

Finally, for some types of data, such as single-cell ATAC-seq, selecting a number of top features is ineffective due to the high sparsity of the signal. In such contexts, rather than _selecting_ features we recommend to use the alternative approach of _aggregating_ similar features (with ‘aggregateFeatures=TRUE‘), which strongly improves accuracy. See the vignette for more detail.

Value

The `sce` object with several additional colData columns, in particular ‘scDblFinder.score‘ (the final score used) and ‘scDblFinder.class‘ (whether the cell is called as 'doublet' or 'singlet'). See `vignette("scDblFinder")` for more details; for alternative return values, see the ‘returnType‘ argument.

Examples

```
library(SingleCellExperiment)
sce <- mockDoubletSCE()
sce <- scDblFinder(sce)
table(truth=sce$type, call=sce$scDblFinder.class)
```

`selFeatures` *selFeatures*

Description

Selects features based on cluster-wise expression or marker detection, or a combination.

Usage

```
selFeatures(
  sce,
  clusters = NULL,
  nfeatures = 1000,
  propMarkers = 0,
  FDR.max = 0.05
)
```

Arguments

`sce`	A `SummarizedExperiment-class`, `SingleCellExperiment-class` with a 'counts' assay.
`clusters`	Optional cluster assignments. Should either be a vector of labels for each cell.
`nfeatures`	The number of features to select.
`propMarkers`	The proportion of features to select from markers (rather than on the basis of high expression). Ignored if ‘clusters‘ isn't given.
`FDR.max`	The maximum marker binom FDR to be included in the selection. (see `findMarkers`).

Value

A vector of feature (i.e. row) names.

Examples

```
sce <- mockDoubletSCE()
selFeatures(sce, clusters=sce$cluster, nfeatures=5)
```

TFIDF *TFIDF*

Description

The Term Frequency - Inverse Document Frequency (TF-IDF) normalization, as implemented in Stuart & Butler et al. 2019.

Usage

```
TFIDF(x, sf = 10000)
```

Arguments

x	The matrix of occurrences
sf	Scaling factor

Value

An array of same dimensions as ‘x‘

Examples

```
m <- matrix(rpois(500,1),nrow=50)
m <- TFIDF(m)
```

Index