Package 'DropletUtils'

October 16, 2019

Version 1.4.3

Date 2019-08-06

Title Utilities for Handling Single-Cell Droplet Data

Depends SingleCellExperiment

Imports S4Vectors, BiocParallel, Rcpp, Matrix, methods, utils, stats, edgeR, rhdf5, HDF5Array, R.utils, dqrng

Suggests testthat, beachmat, knitr, BiocStyle, rmarkdown

biocViews ImmunoOncology, SingleCell, Sequencing, RNASeq, GeneExpression, Transcriptomics, DataImport, Coverage

Description Provides a number of utility functions for handling single-cell (RNA-seq) data from droplet technologies such as 10X Genomics. This includes data loading from count matrices or molecule information files, identification of cells from empty droplets, removal of barcode-swapped pseudo-cells, and downsampling of the count matrix.

License GPL-3

NeedsCompilation yes

VignetteBuilder knitr

LinkingTo Rcpp, beachmat, Rhdf5lib, BH, dqrng

SystemRequirements C++11

RoxygenNote 6.1.1

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

git_branch RELEASE_3_9

git_last_commit c74dbae

git_last_commit_date 2019-08-06

Date/Publication 2019-10-15

Author Aaron Lun [aut, cre],
Jonathan Griffiths [ctb],
Davis McCarthy [ctb]

Maintainer Aaron Lun <infinite.monkeys.with.keyboards@gmail.com>

R topics documented:

`barcodeRanks` *Calculate barcode ranks*

Description

Compute barcode rank statistics and identifry the knee and inflection points on the total count curve.

Usage

```
barcodeRanks(m, lower=100, fit.bounds=NULL, df=20, ...)
```

Arguments

`m`	A real sparse matrix object, either a dgTMatrix or dgCMatrix. Columns represent barcoded droplets, rows represent genes.
`lower`	A numeric scalar specifying the lower bound on the total UMI count, at or below which all barcodes are assumed to correspond to empty droplets.
`fit.bounds`	A numeric vector of length 2, specifying the lower and upper bouunds on the total UMI count for spline fitting.
`df, ...`	Further arguments to pass to `smooth.spline`.

Details

Analyses of droplet-based scRNA-seq data often show a plot of the log-total count against the log-rank of each barcode, where the highest ranks have the largest totals. This is equivalent to a transposed empirical cumulative density plot with log-transformed axes, which focuses on the barcodes with the largest counts. The `barcodeRanks` function will compute these ranks for all barcodes. Barcodes with the same total count receive the same average rank to avoid problems with discrete runs of the same total.

The function will also identify a number of interesting points on the curve for downstream use, namely the inflection and knee points. Both of these points correspond to a sharp transition between two components of the total count distribution, presumably reflecting the difference between empty droplets with little RNA and cell-containing droplets with much more RNA.

- The inflection point is computed as the point on the rank/total curve where the first derivative is minimized. The derivative is computed directly from all points on the curve with total counts greater than `lower`. This avoids issues with erratic behaviour of the curve at lower totals.
- The knee point is defined as the point on the curve where the signed curvature is minimized. This requires calculation of the second derivative, which is much more sensitive to noise in the curve. To overcome this, a smooth spline is fitted to the log-total counts against the log-rank using the `smooth.spline` function. Derivatives are then calculated from the fitted spline using `predict`.

We supply a default setting of `df` to avoid overfitting the spline, as this results in unstability in the higher derivatives (and thus the curvature). `df` and other arguments to `smooth.spline` can be tuned if the estimated knee point is not at an appropriate location. We also restrict the fit to lie within the bounds defined by `fit.bounds` to focus on the region containing the knee point. This allows us to obtain an accurate fit with low `df`, rather than attempting to model the entire curve.

If `fit.bounds` is not specified, the lower bound is automatically set to the inflection point, which should lie after the knee point. The upper bound is set to the point at which the first derivative is closest to zero, i.e., the "plateau" region before the knee point. Note that only points with total counts above `lower` will be considered, regardless of how `fit.bounds` is defined.

Value

A DataFrame where each row corresponds to a column of `m`, and containing the following fields:

`rank`: Numeric, the rank of each barcode (averaged across ties).

`total`: Numeric, the total counts for each barcode.

`fitted`: Numeric, the fitted value from the spline for each barcode. This is `NA` for points with `x` outside of `fit.bounds`.

The metadata contains `knee`, a numeric scalar containing the total count at the knee point; and `inflection`, a numeric scalar containing the total count at the inflection point.

Author(s)

Aaron Lun

See Also

`emptyDrops`

Examples

```
# Mocking up some data:
set.seed(2000)
my.counts <- DropletUtils:::simCounts()

# Computing barcode rank statistics:
br.out <- barcodeRanks(my.counts)
names(br.out)

# Making a plot.
plot(br.out$rank, br.out$total, log="xy", xlab="Rank", ylab="Total")
o <- order(br.out$rank)
lines(br.out$rank[o], br.out$fitted[o], col="red")
abline(h=metadata(br.out)$knee, col="dodgerblue", lty=2)
```

```
abline(h=metadata(br.out)$inflection, col="forestgreen", lty=2)
legend("bottomleft", lty=2, col=c("dodgerblue", "forestgreen"),
    legend=c("knee", "inflection"))
```

`defaultDrops` *Call cells from number of UMIs*

Description

Call cells according to the number of UMIs associated with each barcode, as implemented in CellRanger.

Usage

```
defaultDrops(m, expected=3000, upper.quant=0.99, lower.prop=0.1)
```

Arguments

`m`	A real sparse matrix object, either a dgTMatrix or dgCMatrix. Columns represent barcoded droplets, rows represent cells. The matrix should correspond to an individual sample.
`expected`	A numeric scalar specifying the expected number of cells in this sample, as specified in the call to CellRanger.
`upper.quant`	A numeric scalar between 0 and 1 specifying the quantile of the top `expected` barcodes to consider for the first step of the algorithm
`lower.prop`	A numeric scalar between 0 and 1 specifying the fraction of molecules of the `upper.quant` quantile result that a barcode must exceed to be called as a cell

Details

The `defaultDrops` function will call cells based on library size similarly to the CellRanger software suite from 10X Genomics. Default arguments correspond to an exact reproduction of CellRanger's algorithm, where the number of expected cells was also left at CellRanger default value.

The method computes the `upper.quant` quantile of the top `expected` barcodes, ordered by decreasing number of UMIs. Any barcodes containing more molecules than `lower.prop` times this quantile is considered to be a cell, and is retained for further analysis.

This method may be vulnerable to calling very well-captured background RNA as cells, or missing real cells with low RNA content. See `?emptyDrops` for an alternative approach for cell calling.

Value

`defaultDrops` will return a logical vector of length `ncol(m)`. Each element of the vector reports whether each column of `m` was called as a cell.

Author(s)

Jonathan Griffiths

References

10X Genomics (2017). Cell Ranger Algorithms Overview. https://support.10xgenomics.com/single-cell-gene-expression/software/pipelines/latest/algorithms/overview

See Also

emptyDrops

Examples

```
# Mocking up some data:
set.seed(0)
my.counts <- DropletUtils:::simCounts()

# Identify likely cell-containing droplets.
called <- defaultDrops(my.counts)
table(called)

# Get matrix of called cells.
cell.counts <- my.counts[, called]
```

---

downsampleMatrix *Downsample a count matrix*

Description

Downsample a count matrix to a desired proportion for each cell.

Usage

```
downsampleMatrix(x, prop, bycol=TRUE)
```

Arguments

x	A numeric matrix of counts.
prop	A numeric scalar or, if bycol=TRUE, a vector of length ncol(x). All values should lie in [0, 1] specifying the downsampling proportion for the matrix or for each cell.
bycol	A logical scalar indicating whether downsampling should be performed on a column-by-column basis.

Details

Given multiple batches of very different sequencing depths, it can be beneficial to downsample the deepest batches to match the coverage of the shallowest batches. This avoids differences in technical noise that can drive clustering by batch.

If bycol=TRUE, sampling without replacement is performed on the count vector for each cell. This yields a new count vector where the total is equal to prop times the original total count. Each count in the returned matrix is guaranteed to be smaller than the original value in x. Different proportions can be specified for different cells by setting prop to a vector.

If bycol=FALSE, downsampling without replacement is performed on the entire matrix. This yields a new matrix where the total count across all cells is equal to prop times the original total. The new total count for each cell may not be exactly equal to prop times the original value, which may or may not be more appropriate than bycol=TRUE for particular applications.

Technically, downsampling on the reads with downsampleReads is more appropriate as it recapitulates the effect of differences in sequencing depth per cell. However, in practice, the aim is to obtain

cells that have similar total counts across batches, for which downsampling on the UMI counts is a more direct approach.

Note that this function was originally implemented in the **scater** package as `downsampleCounts`.

Value

A numeric matrix of downsampled counts, of the same type as `x`.

Author(s)

Aaron Lun

See Also

`downsampleReads`

Examples

```
example(read10xCounts)
downsampled <- downsampleMatrix(counts(sce10x), prop = 0.5)
```

---

downsampleReads *Downsample reads in a 10X Genomics dataset*

Description

Generate a UMI count matrix after downsampling reads from the molecule information file produced by CellRanger for 10X Genomics data.

Usage

```
downsampleReads(sample, prop, barcode.length=NULL, bycol=FALSE)
```

Arguments

`sample`	A string containing the path to the molecule information HDF5 file.
`barcode.length`	An integer scalar specifying the length of the cell barcode, see `read10xMolInfo`.
`prop`	A numeric scalar or, if `bycol=TRUE`, a vector of length `ncol(x)`. All values should lie in [0, 1] specifying the downsampling proportion for the matrix or for each cell.
`bycol`	A logical scalar indicating whether downsampling should be performed on a column-by-column basis.

Details

This function downsamples the reads for each molecule by the specified `prop`, using the information in `sample`. It then constructs a UMI count matrix based on the molecules with non-zero read counts. The aim is to eliminate differences in technical noise that can drive clustering by batch, as described in `downsampleMatrix`.

Subsampling the reads with `downsampleReads` recapitulates the effect of differences in sequencing depth per cell. This provides an alternative to downsampling with the CellRanger `aggr` function or subsampling with the 10X Genomics R kit. Note that this differs from directly subsampling the UMI count matrix with `downsampleMatrix`.

If `bycol=FALSE`, downsampling without replacement is performed on all reads from the entire dataset. The total number of reads for each cell after downsampling may not be exactly equal to `prop` times the original value. Note that this is the more natural approach and is the default, which differs from the default used in `downsampleMatrix`.

If `bycol=TRUE`, sampling without replacement is performed on the reads for each cell. The total number of reads for each cell after downsampling is guaranteed to be `prop` times the original total (rounded to the nearest integer). Different proportions can be specified for different cells by setting `prop` to a vector, where each proportion corresponds to a cell/GEM combination in the order returned by `get10xMolInfoStats`.

Value

A numeric sparse matrix containing the downsampled UMI counts for each gene (row) and barcode (column).

Author(s)

Aaron Lun

See Also

`downsampleMatrix`, `read10xMolInfo`

Examples

```
# Mocking up some 10X HDF5-formatted data.
out <- DropletUtils:::sim10xMolInfo(tempfile(), nsamples=1)

# Downsampling by the reads.
downsampleReads(out, barcode.length=4, prop=0.5)
```

---

emptyDrops *Identify empty droplets*

Description

Distinguish between droplets containing cells and ambient RNA in a droplet-based single-cell RNA sequencing experiment.

Usage

```
testEmptyDrops(m, lower=100, niters=10000, test.ambient=FALSE,
    ignore=NULL, alpha=NULL, BPPARAM=SerialParam())

emptyDrops(m, lower=100, retain=NULL, barcode.args=list(), ...)
```

Arguments

m	A numeric matrix object - usually a dgTMatrix or dgCMatrix - containing droplet data *prior to any filtering or cell calling*. Columns represent barcoded droplets, rows represent genes.
lower	A numeric scalar specifying the lower bound on the total UMI count, at or below which all barcodes are assumed to correspond to empty droplets.
niters	An integer scalar specifying the number of iterations to use for the Monte Carlo p-value calculations.
test.ambient	A logical scalar indicating whether results should be returned for barcodes with totals less than or equal to lower.
ignore	A numeric scalar specifying the lower bound on the total UMI count, at or below which barcodes will be ignored (see Details for how this differs from lower).
alpha	A numeric scalar specifying the scaling parameter for the Dirichlet-multinomial sampling scheme.
BPPARAM	A BiocParallelParam object indicating whether parallelization should be used to compute p-values.
retain	A numeric scalar specifying the threshold for the total UMI count above which all barcodes are assumed to contain cells.
barcode.args	Further arguments to pass to barcodeRanks.
...	Further arguments to pass to testEmptyDrops.

Value

testEmptyDrops will return a DataFrame with the following components:

Total: Integer, the total UMI count for each barcode.

LogProb: Numeric, the log-probability of observing the barcode's count vector under the null model.

PValue: Numeric, the Monte Carlo p-value against the null model.

Limited: Logical, indicating whether a lower p-value could be obtained by increasing niters.

For barcodes with counts below lower, NA values are returned for all fields if test.ambient=FALSE. This is to ensure that the number of rows in the output DataFrame is identical to ncol(m).

emptyDrops will return a DataFrame like testEmptyDrops, with an additional FDR field.

The metadata of the output DataFrame will contains the ambient profile in ambient, the estimated/specified value of alpha, the specified value of lower and the number of iterations in niters. For emptyDrops, the metadata will also contain the retention threshold in retain.

Details about `testEmptyDrops`

The `testEmptyDrops` function will obtain an estimate of the composition of the ambient pool of RNA based on the barcodes with total UMI counts less than or equal to `lower`. This assumes that a cell-containing droplet would generally have higher total counts than empty droplets containing RNA from the ambient pool. Counts for the low-count barcodes are pooled together, and an estimate of the proportion vector for the ambient pool is calculated using `goodTuringProportions`. The count vector for each barcode above `lower` is then tested for a significant deviation from these proportions.

The null hypothesis is that transcript molecules are included into droplets by multinomial sampling from the ambient profile. For each barcode, the probability of obtaining its count vector based on the null model is computed. Then, `niters` count vectors are simulated from the null model. The proportion of simulated vectors with probabilities lower than the observed multinomial probability for that barcode is used to calculate the p-value. We use this Monte Carlo approach as an exact multinomial p-value is difficult to calculate.

The `ignore` argument can also be set to ignore barcodes with total counts less than or equal to `ignore`. This differs from the `lower` argument in that the ignored barcodes are not necessarily used to compute the ambient profile. Users can interpret `ignore` as the minimum total count required for a barcode to be considered as a potential cell. In contrast, `lower` is the maximum total count below which all barcodes are assumed to be empty droplets.

Details about `emptyDrops`

The `emptyDrops` function combines the results of `testEmptyDrops` with `barcodeRanks` to identify droplets that are likely to contain cells. Barcodes that contain more than `retain` total counts are always retained. This ensures that large cells with profiles that are very similar to the ambient pool are not inadvertently discarded. If `retain` is not specified, it is set to the total count at the knee point detected by `barcodeRanks`. Manual specification of `retain` may be useful if the knee point was not correctly identified in complex log-rank curves. Users can also set `retain=Inf` to disable automatic retention of barcodes with large totals.

The Benjamini-Hochberg correction is also applied to the Monte Carlo p-values to correct for multiple testing. Cells can then be defined by taking all barcodes with significantly non-ambient profiles, e.g., at a false discovery rate of 0.1%. All barcodes with total counts above `K` (or `retain`) are assigned p-values of zero *during correction*, reflecting our assumption that they are true positives. This ensures that their Monte Carlo p-values do not affect the correction of other genes, and also means that they will have FDR values of zero. Nonetheless, their original Monte Carlo p-values are still reported in the output.

Handling overdispersion

If `alpha` is set to a positive number, sampling is assumed to follow a Dirichlet-multinomial (DM) distribution. The parameter vector of the DM distribution is defined as the estimated ambient profile scaled by `alpha`. Smaller values of `alpha` model overdispersion in the counts, due to dependencies in sampling between molecules. If `alpha=NULL`, a maximum likelihood estimate is obtained from the count profiles for all barcodes with totals less than or equal to `lower`. If `alpha=Inf`, the sampling of molecules is modelled with a multinomial distribution.

Users can check whether the model is suitable by extracting the p-values for all barcodes with `test.ambient=TRUE`. Under the null hypothesis, the p-values for presumed ambient barcodes (i.e., with total counts below `lower`) should be uniformly distributed. Skews in the p-value distribution are indicative of an inaccuracy in the model and/or its estimates (of `alpha` or the ambient profile).

Author(s)

Aaron Lun

References

Lun A, Riesenfeld S, Andrews T, Dao TP, Gomes T, participants in the 1st Human Cell Atlas Jamboree, Marioni JC (2018). Distinguishing cells from empty droplets in droplet-based single-cell RNA sequencing data. *biorXiv*.

Phipson B, Smyth GK (2010). Permutation P-values should never be zero: calculating exact P-values when permutations are randomly drawn. *Stat. Appl. Genet. Mol. Biol.* 9:Article 39.

See Also

barcodeRanks, defaultDrops

Examples

```
# Mocking up some data:
set.seed(0)
my.counts <- DropletUtils:::simCounts()

# Identify likely cell-containing droplets.
out <- emptyDrops(my.counts)
out

is.cell <- out$FDR <= 0.01
sum(is.cell, na.rm=TRUE)

# Check if p-values are lower-bounded by 'niters'
# (increase 'niters' if any Limited==TRUE and Sig==FALSE)
table(Sig=is.cell, Limited=out$Limited)
```

---

encodeSequences *Encode nucleotide sequences*

Description

Encode short nucleotide sequences into integers with a 2-bit encoding.

Usage

```
encodeSequences(sequences)
```

Arguments

sequences	A character vector of short nucleotide sequences, e.g., UMIs or cell barcodes.

Details

Each pair of bits encodes a nucleotide - 00 is A, 01 is C, 10 is G and 11 is T. The least significant byte contains the 3'-most nucleotides, and the remaining bits are set to zero. Thus, the sequence "CGGACT" is converted to the binary form:

```
01 10 10 00 01 11
```

... which corresponds to the integer 1671.

A consequence of R's use of 32-bit integers means that no element of `sequences` can be more than 15 nt long. Otherwise, integer overflow will occur.

Value

An integer vector containing the encoded sequences.

Author(s)

Aaron Lun

References

10X Genomics (2017). Molecule info. https://support.10xgenomics.com/single-cell-gene-expression/software/pipelines/latest/output/molecule_info

Examples

```
encodeSequences("CGGACT")
```

---

get10xMolInfoStats *Get 10x cell statistics*

Description

Compute some basic per-cell statistics from the 10x molecule information file.

Usage

```
get10xMolInfoStats(sample, barcode.length=NULL)
```

Arguments

`sample`	A string containing the path to the molecule information HDF5 file.
`barcode.length`	An integer scalar specifying the length of the cell barcode, see `read10xMolInfo`.

Value

A DataFrame containing one row per cell library, with the fields:

`cell`: Character, the cell barcode.

`gem_group`: Integer, the GEM group.

`num.umis`: Integer, the number of UMIs assigned to this cell barcode/GEM group combination.

`num.reads`: Integer, the number of reads for this combination.

`num.genes`: Integer, the number of detected genes.

Author(s)

Aaron Lun

See Also

read10xMolInfo

Examples

```
# Mocking up some 10X HDF5-formatted data.
out <- DropletUtils:::sim10xMolInfo(tempfile())

get10xMolInfoStats(out)
```

---

makeCountMatrix *Make a count matrix*

Description

Construct a count matrix from per-molecule information, typically the cell and gene of origin.

Usage

```
makeCountMatrix(gene, cell, all.genes=NULL, all.cells=NULL, value=NULL)
```

Arguments

gene	An integer or character vector specifying the gene to which each molecule was assigned.
cell	An integer or character vector specifying the cell to which each molecule was assigned.
all.genes	A character vector containing the names of all genes in the dataset.
all.cells	A character vector containing the names of all cells in the dataset.
value	A numeric vector containing values for each molecule.

Details

Each element of the vectors gene, cell and (if specified) value contain information for a single transcript molecule. Each entry of the output matrix corresponds to a single gene and cell combination. If multiple molecules are present with the same combination, their values in value are summed together, and the sum is used as the entry of the output matrix.

If value=NULL, it will default to a vector of all 1's. Each entry of the output matrix represents the number of molecules with the corresponding combination, i.e., UMI counts. Users can pass other metrics such as the number of reads covering each molecule. This would yield a read count matrix rather than a UMI count matrix.

If all.genes is not specified, it is kept as NULL for integer gene. Otherwise, it is defined as the sorted unique values of character gene. The same occurs for cell and all.cells.

If gene is integer, its values should be positive and no greater than length(all.genes) if all.genes!=NULL. If gene is character, its values should be a subset of those in all.genes. The same requirements apply to cell and all.cells.

Value

A sparse matrix where rows are genes, columns are cells and entries are the sum of `value` for each gene/cell combination. Rows and columns are named if the `gene` or `cell` are character or if `all.genes` or `all.cells` are specified.

Author(s)

Aaron Lun

See Also

`read10xMolInfo`

Examples

```
nmolecules <- 100
gene.id <- sample(LETTERS, nmolecules, replace=TRUE)
cell.id <- sample(20, nmolecules, replace=TRUE)
makeCountMatrix(gene.id, cell.id)
```

`read10xCounts` *Load in data from 10x experiment*

Description

Creates a SingleCellExperiment from the CellRanger output directories for 10X Genomics data.

Usage

```
read10xCounts(samples, col.names=FALSE, type=c("auto", "sparse", "HDF5"),
    version=c("auto", "2", "3"), genome=NULL)
```

Arguments

`samples`	A character vector containing one or more directory names, each corresponding to a 10X sample. Each directory should contain a matrix file, a gene/feature annotation file, and a barcode annotation file. Alternatively, strings may contain a path to a HDF5 file in the sparse matrix format generated by 10X. These can be mixed with directory names when `type="auto"`.
`col.names`	A logical scalar indicating whether the columns of the output object should be named with the cell barcodes.
`type`	String specifying the type of 10X format to read data from.
`version`	String specifying the version of the 10X format to read data from.
`genome`	String specifying the genome if `type="HDF5"` and `version='2'`.

Details

This function was originally developed from the `Read10X` function from the **Seurat** package. It was then taken from the `read10xResults` implementation in the **scater** package.

If `type="auto"`, the format of the input file is automatically detected for each `samples` based on whether it ends with `".h5"`. If so, `type` is set to `"HDF5"`; otherwise it is set to `"sparse"`.

- If `type="sparse"`, count data are loaded as a dgCMatrix object. This is a conventional column-sparse compressed matrix format produced by the CellRanger pipeline.
- If `type="HDF5"`, count data are assumed to follow the 10X sparse HDF5 format for large data sets. It is loaded as a TENxMatrix object, which is a stub object that refers back to the file in `samples`. Users may need to set `genome` if it cannot be automatically determined when `version="2"`.

CellRanger 3.0 introduced a major change in the format of the output files for both `types`. If `version="auto"`, the version of the format is automatically detected from the supplied paths. For `type="sparse"`, this is based on whether there is a `"features.tsv.gz"` file in the directory. For `type="HDF5"`, this is based on whether there is a top-level `"matrix"` group with a `"matrix/features"` subgroup in the file.

Matrices are combined by column if multiple `samples` were specified. This will throw an error if the gene information is not consistent across `samples`.

If `col.names=TRUE` and `length(sample)==1`, each column is named by the cell barcode. For multiple samples, the columns are unnamed to avoid problems with non-unique barcodes across samples.

Note that user-level manipulation of sparse matrices requires loading of the **Matrix** package. Otherwise, calculation of `rowSums`, `colSums`, etc. will result in errors.

Value

A SingleCellExperiment object containing count data for each gene (row) and cell (column) across all `samples`.

- Row metadata will contain the fields `"ID"` and `"Symbol"`. The former is the gene identifier (usually Ensembl), while the latter is the gene name. If `version="3"`, it will also contain the `"Type"` field specifying the type of feature (e.g., gene or antibody).
- Column metadata will contain the fields `"Sample"` and `"Barcode"`. The former contains the value of `samples` from which each column was obtained. The latter refers to the cell barcode sequence and GEM group for each cell library.
- Rows are named with the gene identifier. Columns are named with the cell barcode in certain settings, see Details.
- The assays will contain a single `"counts"` matrix, containing UMI counts for each gene in each cell. Note that the matrix representation will depend on the format of the `samples`, see Details.

Author(s)

Davis McCarthy, with modifications from Aaron Lun

References

Zheng GX, Terry JM, Belgrader P, and others (2017). Massively parallel digital transcriptional profiling of single cells. *Nat Commun* 8:14049.

10X Genomics (2017). Gene-Barcode Matrices. https://support.10xgenomics.com/single-cell-gene-expressi software/pipelines/2.2/output/matrices

10X Genomics (2018). Feature-Barcode Matrices. https://support.10xgenomics.com/single-cell-gene-expres software/pipelines/latest/output/matrices

10X Genomics (2018). HDF5 Gene-Barcode Matrix Format. https://support.10xgenomics. com/single-cell-gene-expression/software/pipelines/2.2/advanced/h5_matrices

10X Genomics (2018). HDF5 Feature Barcode Matrix Format. https://support.10xgenomics. com/single-cell-gene-expression/software/pipelines/latest/advanced/h5_matrices

See Also

write10xCounts

Examples

```
# Mocking up some 10X genomics output.
example(write10xCounts)

# Reading it in.
sce10x <- read10xCounts(tmpdir)

# Column names are dropped with multiple 'samples'.
sce10x2 <- read10xCounts(c(tmpdir, tmpdir))
```

---

read10xMolInfo *Read the 10X molecule information file*

Description

Extract relevant fields from the molecule information HDF5 file, produced by CellRanger for 10X Genomics data.

Usage

```
read10xMolInfo(sample, barcode.length=NULL, keep.unmapped=FALSE, get.cell=TRUE,
    get.umi=TRUE, get.gem=TRUE, get.gene=TRUE, get.reads=TRUE,
    version=c("auto", "2", "3"))
```

Arguments

sample	A string containing the path to the molecule information HDF5 file.
barcode.length	An integer scalar specifying the length of the cell barcode. Only relevant when version="2".
keep.unmapped	A logical scalar indicating whether unmapped molecules should be reported.
get.cell, get.umi, get.gem, get.gene, get.reads	Logical scalar indicating whether the corresponding field should be extracted.
version	String specifying the version of the 10X molecule information format to read data from.

Details

Molecules that were not assigned to any gene have gene set to `length(genes)+1`. By default, these are removed when `keep.unmapped=FALSE`.

CellRanger 3.0 introduced a major change in the format of the molecule information files. When `version="auto"`, the function will attempt to determine the version format of the file. This can also be user-specified by setting `version` explicitly.

For files produced by version 2.2 of the CellRanger software, the length of the cell barcode is not given. Instead, the barcode length is automatically inferred if `barcode.length=NULL` and `version="2"`. Currently, version 1 of the 10X chemistry uses 14 nt barcodes, while version 2 uses 16 nt barcodes.

Setting any of the `get.*` arguments will (generally) avoid extraction of the corresponding field. This can improve efficiency if that field is not necessary for further analysis. Aside from the missing field, the results are guaranteed to be identical, i.e., same order and number of rows. Note that some fields must be loaded to yield similar results, e.g., gene IDs will always be loaded to remove unmapped reads if `keep.unmapped=FALSE`.

Value

A list is returned containing two elements. The first element is named `data` and is a DataFrame where each row corresponds to a single transcript molecule. The fields are as follows:

`barcode`: Character, the cell barcode for each molecule.

`umi`: Integer, the processed UMI barcode in 2-bit encoding.

`gem_group`: Integer, the GEM group.

`gene`: Integer, the index of the gene to which the molecule was assigned. This refers to an entry in the `genes` vector, see below.

`reads`: Integer, the number of reads mapped to this molecule.

The field will not be present in the DataFrame if the corresponding `get.*` argument is `FALSE`,

The second element of the list is named `genes` and is a character vector containing the names of all genes in the annotation. This contains the names of the various entries of `gene` for the individual molecules.

Author(s)

Aaron Lun, based on code by Jonathan Griffiths

References

Zheng GX, Terry JM, Belgrader P, and others (2017). Massively parallel digital transcriptional profiling of single cells. *Nat Commun* 8:14049.

10X Genomics (2017). Molecule info. https://support.10xgenomics.com/single-cell-gene-expression/software/pipelines/2.2/output/molecule_info

10X Genomics (2018). Molecule info. https://support.10xgenomics.com/single-cell-gene-expression/software/pipelines/latest/output/molecule_info

See Also

`makeCountMatrix`

Examples

```
# Mocking up some 10X HDF5-formatted data.
out <- DropletUtils:::sim10xMolInfo(tempfile())

# Reading the resulting file.
read10xMolInfo(out)
```

---

swappedDrops *Clean barcode-swapped droplet data*

Description

Remove the effects of barcode swapping on droplet-based single-cell RNA-seq data, specifically 10X Genomics datasets.

Usage

```
swappedDrops(samples, barcode.length=NULL, ...)

removeSwappedDrops(cells, umis, genes, nreads, ref.genes, min.frac=0.8,
    get.swapped=FALSE, get.diagnostics=FALSE, hdf5.out=TRUE)
```

Arguments

samples	A character vector containing paths to the molecule information HDF5 files, produced by CellRanger for 10X Genomics data. Each file corresponds to one sample in a multiplexed pool.
barcode.length	An integer scalar specifying the length of the cell barcode, see read10xMolInfo.
...	Further arguments to be passed to removeSwappedDrops.
cells	A list of character vectors containing cell barcodes. Each vector corresponds to one sample in a multiplexed pool, and each entry of the vector corresponds to one molecule.
umis	A list of integer vectors containing encoded UMI sequences, organized as described for cells. See ?encodeSequences to convert sequences to integers.
genes	A list of integer vectors specifying the gene indices, organized as described for cells. Each index should refer to an element of ref.genes.
nreads	A list of integer vectors containing the number of reads per molecule, organized as described for cells.
ref.genes	A character vector containing the names or symbols of all genes.
min.frac	A numeric scalar specifying the minimum fraction of reads required for a swapped molecule to be assigned to a sample.
get.swapped	A logical scalar indicating whether the UMI counts corresponding to swapped molecules should also be returned.
get.diagnostics	A logical scalar indicating whether to return the number of reads for each swapped molecule in each sample.
hdf5.out	A logical scalar indicating whether the diagnotic matrix should be returned as a HDF5Matrix.

Details

Barcode swapping on the Illumina sequencer occurs when multiplexed samples undergo PCR re-amplification on the flow cell by excess primer with different barcodes. This results in sequencing of the wrong sample barcode and molecules being assigned to incorrect samples after debarcoding. With droplet data, there is the opportunity to remove such effects based on the combination of gene, UMI and cell barcode for each observed transcript molecule. It is very unlikely that the same combination will arise from different molecules in multiple samples. Thus, observation of the same combination across multiple samples is indicative of barcode swapping.

We can remove swapped molecules based on the number of reads assigned to each gene-UMI-barcode combination. From the total number of reads assigned to that combination, the fraction of reads in each sample is calculated. The sample with the largest fraction that is greater than `min.frac` is defined as the putative sample of origin to which the molecule is assigned. This assumes that the swapping rate is low, so the sample of origin for a molecule should contain the majority of the reads. In other all samples, reads for the combination are assumed to derive from swapping and do not contribute to the count matrix. Setting `min.frac=1` will effectively remove all molecules that appear in multiple samples. We do not recommend setting `min.frac` lower than 0.5.

If `diagnostics=TRUE`, a diagnostics matrix is returned containing the number of reads per gene-UMI-barcode combination in each sample. Each row corresponds to a combination and each column corresponds to a sample. This can be useful for examining the level of swapping across samples on a molecule-by-molecule basis, though for the sake of memory, the actual identity of the molecules is not returned. By default, the matrix is returned as a HDF5Matrix, which reduces memory usage and avoids potential issues with integer overflow. If `hdf5.out=FALSE`, a sparse matrix is returned instead, which is faster but uses more memory.

`swappedDrops` is a wrapper around `removeSwappedDrops` that extracts the relevant data from the 10X Genomics molecule information file. For other types of droplet-based data, it may be more convenient to call `removeSwappedDrops` directly.

Value

A list is returned with the `cleaned` entry, itself a list of sparse matrices is returned. Each matrix corresponds to a sample and contains the UMI count for each gene (row) and cell barcode (column) after removing swapped molecules. All cell barcodes that were originally observed are reported as columns, though note that it is possible for some barcodes to contain no counts.

If `get.swapped=TRUE`, a `swapped` entry is returned in the top-level list. This is a list containing sample-specific sparse matrices of UMI counts corresponding to the swapped molecules. Adding the cleaned and swapped matrices for each sample should yield the total UMI count prior to removal of swapped molecules.

If `get.diagnostics=TRUE`, the top-level list will also contain an additional `diagnostics` matrix.

Format of the molecule information file

`swappedDrops` makes a few assumptions about the nature of the data in each molecule information file. These are necessary to simplify downstream processing and are generally acceptable in most cases.

Each molecule information file should contain data from only a single 10X run. Users should *not* combine multiple samples into a single molecule information file. The function will emit a warning upon detecting multiple GEM groups from any molecule information file. Molecules with different GEMs will not be recognised as coming from a different sample, though they will be recognised as being derived from different cell-level libraries.

In files produced by CellRanger version 3.0, an additional per-molecule field is present indicating the (c)DNA library from which the molecule was derived. Library preparation can be performed separately for different features (e.g., antibodies, CRISPR tags) such that one 10X run can contain data from multiple libraries. This allows for arbitrarily complicated multiplexing schemes - for example, gene expression libraries might be multiplexed together across one set of samples, while the antibody-derived libraries might be multiplexed across another *different* set of samples. For simplicity, we assume that multiplexing was performed across the same set of `samples` for all libraries therein.

Author(s)

Jonathan Griffiths, with modifications by Aaron Lun

References

Griffiths JA, Lun ATL, Richard AC, Bach K, Marioni JC (2018). Detection and removal of barcode swapping in single-cell RNA-seq data. *Nat. Commun.* 9, 1:2667.

See Also

`read10xMolInfo`, `encodeSequences`

Examples

```
# Mocking up some 10x HDF5-formatted data, with swapping.
curfiles <- DropletUtils:::sim10xMolInfo(tempfile(), nsamples=3)

# Obtaining count matrices with swapping removed.
out <- swappedDrops(curfiles)
lapply(out$cleaned, dim)

out <- swappedDrops(curfiles, get.swapped=TRUE, get.diagnostics=TRUE)
names(out)
```

---

write10xCounts *Write count data in the 10x format*

Description

Create a directory containing the count matrix and cell/gene annotation from a sparse matrix of UMI counts, in the format produced by the CellRanger software suite.

Usage

```
write10xCounts(path, x, barcodes=colnames(x), gene.id=rownames(x),
    gene.symbol=gene.id, gene.type="Gene Expression", overwrite=FALSE,
    type=c("auto", "sparse", "HDF5"), genome="unknown", version=c("2", "3"))
```

Arguments

x	A sparse numeric matrix of UMI counts.
path	A string containing the path to the output directory.
barcodes	A character vector of cell barcodes, one per column of x.
gene.id	A character vector of gene identifiers, one per row of x.
gene.symbol	A character vector of gene symbols, one per row of x.
gene.type	A character vector of gene types, expanded to one per row of x. Only used when version="3".
overwrite	A logical scalar specifying whether path should be overwritten if it already exists.
type	String specifying the type of 10X format to save x to.
genome	String specifying the genome for storage when type="HDF5". This can be a character vector with one genome per feature if version="3".
version	String specifying the version of the CellRanger format to produce.

Details

This function will try to automatically detect the desired format based on whether path ends with ".h5". If so, it assumes that path specifies a HDF5 file path and sets type="HDF5". Otherwise it will set type="sparse" under the assumption that path specifies a path to a directory.

Note that there were major changes in the output format for CellRanger version 3.0, to account for non-gene features such as antibody or CRISPR tags. Users can switch to this new format using version="3". See the documentation for "latest" for this new format, otherwise see "2.2" or earlier.

Value

For type="sparse", a directory is produced at path. If version="2", this will contain the files "matrix.mtx", "barcodes.tsv" and "genes.tsv". If version="3", it will instead contain "matrix.mtx.gz", "barcodes.tsv.gz" and "features.tsv.gz".

For type="HDF5", a HDF5 file is produced at path containing data in column-sparse format. If version="2", data are stored in the HDF5 group named genome. If version="3", data are stored in the group "matrix".

A TRUE value is invisibly returned.

Author(s)

Aaron Lun

References

10X Genomics (2017). Gene-Barcode Matrices. https://support.10xgenomics.com/single-cell-gene-expressi
software/pipelines/2.2/output/matrices

10X Genomics (2018). Feature-Barcode Matrices. https://support.10xgenomics.com/single-cell-gene-expres
software/pipelines/latest/output/matrices

10X Genomics (2018). HDF5 Gene-Barcode Matrix Format. https://support.10xgenomics.
com/single-cell-gene-expression/software/pipelines/2.2/advanced/h5_matrices

10X Genomics (2018). HDF5 Feature Barcode Matrix Format. https://support.10xgenomics.
com/single-cell-gene-expression/software/pipelines/latest/advanced/h5_matrices

See Also

read10xCounts

Examples

```
# Mocking up some count data.
library(Matrix)
my.counts <- matrix(rpois(1000, lambda=5), ncol=10, nrow=100)
my.counts <- as(my.counts, "dgCMatrix")
cell.ids <- paste0("BARCODE-", seq_len(ncol(my.counts)))

ngenes <- nrow(my.counts)
gene.ids <- paste0("ENSG0000", seq_len(ngenes))
gene.symb <- paste0("GENE", seq_len(ngenes))

# Writing this to file:
tmpdir <- tempfile()
write10xCounts(tmpdir, my.counts, gene.id=gene.ids,
    gene.symbol=gene.symb, barcodes=cell.ids)
list.files(tmpdir)
```

Index