

Package: presto (via r-universe)

October 1, 2026

Title Fast Functions for Differential Expression Using Wilcoxon and AUC

Version 1.1.0

Description Scalable implementation of the Wilcoxon rank sum test and the area under the receiver operating characteristic curve (also known as auROC or AUC) statistic. Interfaces to dense and sparse matrices, as well as the genomics analysis frameworks 'Seurat' and 'SingleCellExperiment'.

License GPL-3

Encoding UTF-8

Depends R (>= 3.5.0)

LinkingTo Rcpp, RcppArmadillo

Imports Rcpp, data.table, dplyr, methods, tidyr, purrr, tibble, Matrix, rlang, stats, utils

RoxygenNote 7.3.3

Suggests knitr, rmarkdown, testthat, Seurat, SingleCellExperiment, SummarizedExperiment, DelayedArray, broom, BiocStyle, DESeq2

NeedsCompilation yes

VignetteBuilder knitr

Author Ilya Korsunsky [aut] (ORCID: <<https://orcid.org/0000-0003-4848-3948>>), Aparna Nathan [aut], Nghia Millard [aut] (ORCID: <<https://orcid.org/0000-0002-0518-7674>>), Soumya Raychaudhuri [aut] (ORCID: <<https://orcid.org/0000-0002-1901-8265>>), Kamil Slowikowski [cre, ctb] (ORCID: <<https://orcid.org/0000-0002-2843-6370>>), Austin Hartman [ctb] (ORCID: <<https://orcid.org/0000-0001-7278-1852>>)

Maintainer Kamil Slowikowski <kslowikowski@gmail.com>

Config/pak/sysreqs libicu-dev

Repository <https://slowkow.r-universe.dev>

Date/Publication 2026-09-30 13:56:03 UTC

RemoteUrl <https://github.com/cran/presto>
RemoteRef HEAD
RemoteSha 5e323fb82db78c8f5b7edbec2c68e20dcb43a9a8

Contents

collapse_counts	2
compute_hash	4
nnzeroGroups	4
pseudobulk_deseq2	5
pseudobulk_one_vs_all	7
pseudobulk_pairwise	8
pseudobulk_within	10
rank_matrix	11
sumGroups	12
summarize_dge_pairs	13
top_markers	14
top_markers_dds	15
toy_sce	16
toy_seurat	16
wilcoxauc	17

Index	21
--------------	-----------

collapse_counts	<i>Collapse a single-cell count matrix into pseudobulks</i>
-----------------	---

Description

Sums (or averages) the columns of a feature-by-cell count matrix according to one or more cell-metadata columns. The result is a feature-by-pseudobulk matrix where each column pools all cells that share the same combination of metadata values (e.g. all cells from a given donor in a given cluster). The resulting matrix is suitable for bulk-RNA-seq tools such as DESeq2, edgeR, or limma. See the pseudobulk vignette for an end-to-end walkthrough.

Usage

```
collapse_counts(  
  counts_mat,  
  meta_data,  
  varnames,  
  min_cells_per_group = 0,  
  keep_n = FALSE,  
  how = c("sum", "mean")[1]  
)
```

Arguments

counts_mat	Counts matrix. Rows are features (genes), columns are cells. Sparse (dgCMatrix) or dense.
meta_data	data.frame of cell metadata. Must have one row per column of counts_mat.
varnames	Character vector of column names in meta_data that together define the pseudobulk grouping. Each unique combination of values across these columns becomes one pseudobulk sample.
min_cells_per_group	Drop pseudobulks containing fewer than this many cells. Default 0 (keep all).
keep_n	If TRUE, retain the per-pseudobulk cell count as column N in the returned meta_data. Default FALSE.
how	"sum" (default) sums counts across cells in each pseudobulk; "mean" divides by N to give the mean expression per cell. DESeq2 and other count-based regressions expect sums.

Value

A list with two elements:

- counts_mat - feature-by-pseudobulk numeric matrix.
- meta_data - data.frame with one row per pseudobulk containing the columns named in varnames (and N if keep_n = TRUE).

See Also

[pseudobulk_deseq2\(\)](#), [compute_hash\(\)](#)

Examples

```
m <- matrix(sample.int(8, 100*500, replace=TRUE), nrow=100, ncol=500)
rownames(m) <- paste0("G", 1:100)
colnames(m) <- paste0("C", 1:500)
md1 <- sample(c("a", "b"), 500, replace=TRUE)
md2 <- sample(c("c", "d"), 500, replace=TRUE)
df <- data.frame(md1, md2)
data_collapsed <- collapse_counts(m, df, c("md1", "md2"))
head(data_collapsed$counts_mat)
head(data_collapsed$meta_data)
```

compute_hash	<i>Compute a unique integer hash for each row of a data.frame</i>
--------------	---

Description

Treats the supplied columns as a composite key: rows that agree on every value in `vars_use` receive the same integer hash, distinct combinations receive distinct hashes. Used internally by `collapse\_counts\(\)` to identify pseudobulk groups, but exposed for callers who need the same indexing in their own code.

Usage

```
compute_hash(data_df, vars_use)
```

Arguments

<code>data_df</code>	A data.frame.
<code>vars_use</code>	Character vector of column names in <code>data_df</code> . Each column is coerced to factor before hashing.

Value

Integer vector of length `nrow(data_df)`.

See Also

`collapse\_counts\(\)`

nnzeroGroups	<i>Group-wise non-zero counts of a matrix along one axis</i>
--------------	--

Description

For each unique value of the grouping vector `y`, counts the number of non-zero entries among the corresponding rows (or columns) of `X`. Used internally by `wilcoxauc\(\)` to compute the percent-expressed columns (`pct_in`, `pct_out`), but exposed as a fast group-wise reduction primitive for both dense and `dgCMatrix` inputs.

Usage

```
nnzeroGroups(X, y, MARGIN = 2)
```

```
## S3 method for class 'dgCMatrix'
nnzeroGroups(X, y, MARGIN = 2)
```

```
## S3 method for class 'matrix'
nnzeroGroups(X, y, MARGIN = 2)
```

Arguments

<code>X</code>	Numeric matrix or <code>dgCMatrix</code> .
<code>y</code>	Group label vector. Coerced to integer factor codes.
<code>MARGIN</code>	Whether observations are along rows or columns of <code>X</code> . <code>MARGIN = 2</code> (default): observations are rows (<code>length(y) == nrow(X)</code>). <code>MARGIN = 1</code> : observations are columns (<code>length(y) == ncol(X)</code>).

Value

Integer matrix of shape `n_groups` x `n_features`, where entry `(g, j)` is the number of observations in group `g` for which feature `j` is non-zero.

See Also

[sumGroups\(\)](#), [wilcoxauc\(\)](#)

Examples

```
set.seed(42)
exprs <- matrix(rpois(25 * 150, lambda = 2), nrow = 25,
                dimnames = list(paste0("G", 1:25), NULL))
y <- rep(c("A", "B", "C"), each = 50)
nnz_res <- nnzeroGroups(exprs, y, 1)
nnz_res <- nnzeroGroups(t(exprs), y, 2)
```

pseudobulk_deseq2

Pseudobulk differential expression with DESeq2

Description

Runs `DESeq2::DESeq()` on a feature-by-pseudobulk count matrix to identify genes whose expression differs between groups of pseudobulks. Three test designs are supported, selected by mode:

- "one_vs_all" (default) - test each level of the contrast variable against the union of all others. Useful for marker discovery. Dispatched to [pseudobulk_one_vs_all\(\)](#).
- "pairwise" - test every ordered pair of levels separately. Useful for high-confidence markers; combine with [summarize_dge_pairs\(\)](#) to keep the most conservative pair. Dispatched to [pseudobulk_pairwise\(\)](#).
- "within" - split on the first term of `dge_formula` and test the second term within each split level. Useful for condition / case-vs-control comparisons restricted to one cluster at a time. Dispatched to [pseudobulk_within\(\)](#).

Pseudobulk inputs are typically produced by [collapse_counts\(\)](#). Genes with low counts across pseudobulks are filtered out before fitting (controlled by `min_counts_per_sample` and `present_in_min_samples`). See the pseudobulk vignette for an end-to-end walkthrough on a real single-cell dataset.

Usage

```
pseudobulk_deseq2(
  dge_formula,
  meta_data,
  counts_df,
  verbose = TRUE,
  min_counts_per_sample = 10,
  present_in_min_samples = 5,
  collapse_background = TRUE,
  vals_test = NULL,
  mode = c("one_vs_all", "pairwise", "within")[1]
)
```

Arguments

<code>dge_formula</code>	One-sided formula such as <code>~cluster + donor</code> . The first term is treated as the contrast variable in "one_vs_all" and "pairwise" modes, or as the split variable in "within" mode (in which case the second term becomes the contrast variable). Additional terms are kept as covariates in the DESeq2 design.
<code>meta_data</code>	data.frame of pseudobulk metadata. One row per pseudobulk; should contain only the variables used in <code>dge_formula</code> .
<code>counts_df</code>	Feature-by-pseudobulk integer count matrix. Rows are features; columns must align with rows of <code>meta_data</code> .
<code>verbose</code>	Logical. Print progress messages. Default TRUE.
<code>min_counts_per_sample</code>	Minimum count per pseudobulk for a gene to be considered expressed in that pseudobulk. Default 10.
<code>present_in_min_samples</code>	Minimum number of pseudobulks in which a gene must reach <code>min_counts_per_sample</code> to be retained. Default 5.
<code>collapse_background</code>	Used only when <code>mode = "one_vs_all"</code> . If TRUE, background pseudobulks are collapsed across the contrast variable before each DESeq2 fit, which helps when donor representation is unbalanced across clusters. Default TRUE.
<code>vals_test</code>	Character vector of contrast levels to test. If NULL (default), every level of the contrast variable is tested.
<code>mode</code>	One of "one_vs_all" (default), "pairwise", or "within". See description.

Value

A long-form data.frame of DESeq2 results. Columns include the group identifier(s) (group, or group1 / group2 in pairwise mode), feature, and the DESeq2 columns `baseMean`, `log2FoldChange`, `lfcSE`, `stat`, `pvalue`, `padj`.

See Also

[collapse_counts\(\)](#), [top_markers_dds\(\)](#), [summarize_dge_pairs\(\)](#)

Examples

```

if (requireNamespace("DESeq2", quietly = TRUE)) {
  ## 40 genes x 300 cells from 2 clusters across 6 donors
  m <- matrix(sample.int(8, 40 * 300, replace = TRUE), nrow = 40)
  rownames(m) <- paste0("G", 1:40)
  colnames(m) <- paste0("C", 1:300)
  meta <- data.frame(
    cluster = sample(c("a", "b"), 300, replace = TRUE),
    donor = sample(paste0("d", 1:6), 300, replace = TRUE)
  )

  ## collapse cells into per-(cluster, donor) pseudobulks
  dc <- collapse_counts(m, meta, c("cluster", "donor"))

  ## test each cluster against the rest
  res <- pseudobulk_deseq2(
    ~cluster,
    dc$meta_data["cluster"],
    dc$counts_mat,
    verbose = FALSE,
    present_in_min_samples = 1,
    collapse_background = FALSE,
    mode = "one_vs_all"
  )
  head(res)
}

```

pseudobulk_one_vs_all *Pseudobulk DESeq2: one-vs-all contrasts*

Description

For each level of contrast_var, fits a DESeq2 model where that level is the foreground and every other level is pooled as the background. Useful for marker-style differential expression: the return is one set of (log2FoldChange, padj) per gene per group, giving a quick view of what's elevated in each group relative to the rest.

Usage

```

pseudobulk_one_vs_all(
  dge_formula,
  counts_df,
  meta_data,
  contrast_var,
  vals_test,
  collapse_background,
  verbose
)

```

Arguments

dge_formula	One-sided formula such as <code>~cluster + donor</code> . The first term is treated as the contrast variable in "one_vs_all" and "pairwise" modes, or as the split variable in "within" mode (in which case the second term becomes the contrast variable). Additional terms are kept as covariates in the DESeq2 design.
counts_df	Feature-by-pseudobulk integer count matrix. Rows are features; columns must align with rows of meta_data.
meta_data	data.frame of pseudobulk metadata. One row per pseudobulk; should contain only the variables used in dge_formula.
contrast_var	Name of the contrast column in meta_data (the first term of dge_formula).
vals_test	Character vector of contrast levels to test. If NULL (default), every level of the contrast variable is tested.
collapse_background	Used only when mode = "one_vs_all". If TRUE, background pseudobulks are collapsed across the contrast variable before each DESeq2 fit, which helps when donor representation is unbalanced across clusters. Default TRUE.
verbose	Logical. Print progress messages. Default TRUE.

Details

Most users should call `pseudobulk_deseq2()` with mode = "one_vs_all" rather than this function directly; the wrapper handles formula parsing, gene-count filtering, and dispatch. See the pseudobulk vignette for a worked example.

Value

data.frame of DESeq2 results with columns group, feature, baseMean, log2FoldChange, lfcSE, stat, pvalue, padj. Sorted by stat descending within each group.

See Also

`pseudobulk_deseq2()`, `pseudobulk_pairwise()`, `pseudobulk_within()`, `top_markers_dds()`

pseudobulk_pairwise *Pseudobulk DESeq2: pairwise contrasts*

Description

For each ordered pair of contrast levels in vals_test, fits a DESeq2 model on just those two groups of pseudobulks and returns the foreground-vs-background coefficient. This is more conservative than one-vs-all because a marker has to differentiate the foreground from *every* other group, not just the average background. Cost grows as $O(N^2)$ in the number of levels, so subset to the levels of interest first.

Usage

```
pseudobulk_pairwise(
  dge_formula,
  counts_df,
  meta_data,
  contrast_var,
  vals_test,
  verbose,
  min_counts_per_sample,
  present_in_min_samples
)
```

Arguments

dge_formula	One-sided formula such as <code>~cluster + donor</code> . The first term is treated as the contrast variable in "one_vs_all" and "pairwise" modes, or as the split variable in "within" mode (in which case the second term becomes the contrast variable). Additional terms are kept as covariates in the DESeq2 design.
counts_df	Feature-by-pseudobulk integer count matrix. Rows are features; columns must align with rows of meta_data.
meta_data	data.frame of pseudobulk metadata. One row per pseudobulk; should contain only the variables used in dge_formula.
contrast_var	Name of the contrast column in meta_data (the first term of dge_formula).
vals_test	Character vector of contrast levels to test. If NULL (default), every level of the contrast variable is tested.
verbose	Logical. Print progress messages. Default TRUE.
min_counts_per_sample	Minimum count per pseudobulk for a gene to be considered expressed in that pseudobulk. Default 10.
present_in_min_samples	Minimum number of pseudobulks in which a gene must reach min_counts_per_sample to be retained. Default 5.

Details

Most users should call `pseudobulk_deseq2()` with `mode = "pairwise"` rather than this function directly. The companion `summarize_dge_pairs()` collapses the directional pairs to a single row per (gene, group).

Value

data.frame of DESeq2 results with columns `group1`, `group2`, `feature`, `baseMean`, `log2FoldChange`, `lfcSE`, `stat`, `pvalue`, `padj`. Each row reports the test where pseudobulks of group1 are the foreground and pseudobulks of group2 are the background.

See Also

[pseudobulk_deseq2\(\)](#), [pseudobulk_one_vs_all\(\)](#), [pseudobulk_within\(\)](#), [summarize_dge_pairs\(\)](#)

pseudobulk_within	<i>Pseudobulk DESeq2: within-group contrasts</i>
-------------------	--

Description

Splits the pseudobulks by `split_var` and, within each split level, fits a DESeq2 model whose contrast variable is the second term of `dge_formula`. Used to test for an effect (e.g. case vs. control) restricted to one cluster at a time, where pooling across clusters would mix biology and batch.

Usage

```
pseudobulk_within(
  dge_formula,
  counts_df,
  meta_data,
  split_var,
  vals_test,
  verbose,
  min_counts_per_sample,
  present_in_min_samples
)
```

Arguments

<code>dge_formula</code>	One-sided formula such as <code>~cluster + donor</code> . The first term is treated as the contrast variable in "one_vs_all" and "pairwise" modes, or as the split variable in "within" mode (in which case the second term becomes the contrast variable). Additional terms are kept as covariates in the DESeq2 design.
<code>counts_df</code>	Feature-by-pseudobulk integer count matrix. Rows are features; columns must align with rows of <code>meta_data</code> .
<code>meta_data</code>	data.frame of pseudobulk metadata. One row per pseudobulk; should contain only the variables used in <code>dge_formula</code> .
<code>split_var</code>	Name of the column in <code>meta_data</code> to split on (the first term of <code>dge_formula</code>). Each unique level of <code>split_var</code> produces an independent DESeq2 fit.
<code>vals_test</code>	Character vector of contrast levels to test. If NULL (default), every level of the contrast variable is tested.
<code>verbose</code>	Logical. Print progress messages. Default TRUE.
<code>min_counts_per_sample</code>	Minimum count per pseudobulk for a gene to be considered expressed in that pseudobulk. Default 10.
<code>present_in_min_samples</code>	Minimum number of pseudobulks in which a gene must reach <code>min_counts_per_sample</code> to be retained. Default 5.

Details

Two-level contrasts (factor or character) yield the standard `<var>_<level2>_vs_<level1>` Wald coefficient. Three or more levels are integer-encoded and the fit returns an ordinal trend. Most users should call `pseudobulk_deseq2()` with `mode = "within"` rather than this function directly. See the pseudobulk vignette for a worked example using case-vs-control DGE within each cell cluster.

Value

data.frame of DESeq2 results with columns `group` (the value of `split_var`), `feature`, `baseMean`, `log2FoldChange`, `lfcSE`, `stat`, `pvalue`, `padj`.

See Also

`pseudobulk_deseq2()`, `pseudobulk_one_vs_all()`, `pseudobulk_pairwise()`

rank_matrix	<i>Column-wise tied ranks of a matrix</i>
-------------	---

Description

Ranks the entries of each column independently using the average rank for ties, and returns the per-column tie group sizes needed for the Wilcoxon variance correction. Used internally by `wilcoxauc()` (on a transposed input, so that rows become observations) but exposed as a fast standalone ranking primitive for sparse and dense numeric matrices.

Usage

```
rank_matrix(X)

## S3 method for class 'dgCMatrx'
rank_matrix(X)

## S3 method for class 'matrix'
rank_matrix(X)
```

Arguments

`X` Numeric matrix or dgCMatrx.

Value

List with two elements:

- `X_ranked` - matrix with the same shape as `X` containing per-column tied ranks.
- `ties` - list of integer vectors, one per column, giving the sizes of all tie groups encountered in that column. Used by the Wilcoxon statistic to correct for ties.

See Also[wilcoxauc\(\)](#)**Examples**

```
set.seed(42)
exprs <- matrix(rpois(25 * 150, lambda = 2), nrow = 25,
               dimnames = list(paste0("G", 1:25), NULL))
rank_res <- rank_matrix(exprs)
```

sumGroups

*Group-wise sum of a matrix along one axis***Description**

For each unique value of the grouping vector `y`, sums the corresponding rows (or columns) of `X`. Used internally by [wilcoxauc\(\)](#) and [collapse_counts\(\)](#), but exposed as a fast group-wise reduction primitive that works on both dense matrices and `dgCMatrix` sparse inputs.

Usage

```
sumGroups(X, y, MARGIN = 2)

## S3 method for class 'dgCMatrix'
sumGroups(X, y, MARGIN = 2)

## S3 method for class 'matrix'
sumGroups(X, y, MARGIN = 2)
```

Arguments

<code>X</code>	Numeric matrix or <code>dgCMatrix</code> .
<code>y</code>	Group label vector. Coerced to integer factor codes.
<code>MARGIN</code>	Whether observations are along rows or columns of <code>X</code> . <code>MARGIN = 2</code> (default): observations are rows (<code>length(y) == nrow(X)</code>); rows are summed within each group. <code>MARGIN = 1</code> : observations are columns (<code>length(y) == ncol(X)</code>); columns are summed within each group.

Value

Numeric matrix of shape `n_groups` x `n_features`. Row order matches the integer order of `factor(y)`.

See Also[nnzeroGroups\(\)](#), [wilcoxauc\(\)](#)

Examples

```
set.seed(42)
exprs <- matrix(rpois(25 * 150, lambda = 2), nrow = 25,
  dimnames = list(paste0("G", 1:25), NULL))
y <- rep(c("A", "B", "C"), each = 50)
sumGroups_res <- sumGroups(exprs, y, 1)
sumGroups_res <- sumGroups(t(exprs), y, 2)
```

summarize_dge_pairs	<i>Summarize directional pairwise DGE results</i>
---------------------	---

Description

Collapses the long-form output of `pseudobulk_deseq2(mode = "pairwise")` to a single row per (group, gene) by selecting one comparison per gene-group pair. With mode = "min" it keeps the worst comparison (most conservative — the gene must beat *every* other group to have a high statistic). With mode = "max" it keeps the best (most permissive — the gene need only beat one).

Usage

```
summarize_dge_pairs(dge_res, mode = c("min", "max")[1])
```

Arguments

dge_res	data.frame of pairwise results from <code>pseudobulk_deseq2()</code> with mode = "pairwise". Must have columns group1, group2, feature, and stat.
mode	"min" (default) keeps the comparison with the smallest stat for each (group, gene); "max" keeps the largest.

Value

data.frame with one row per (group, gene), sorted by stat descending within each group. The group2 column is dropped and group1 is renamed to group.

See Also

[pseudobulk_deseq2\(\)](#), [pseudobulk_pairwise\(\)](#)

top_markers

*Top markers per group from wilcoxauc results***Description**

Filters and ranks the long-form output of `wilcoxauc()` to give the most distinguishing features per group. The filter arguments combine multiplicatively, then the top *n* features per group are kept by descending auc and pivoted into wide form. Counterpart to `top_markers_dds()` for DESeq2-based pseudobulk results.

Usage

```
top_markers(
  res,
  n = 10,
  auc_min = 0,
  pval_max = 1,
  padj_max = 1,
  pct_in_min = 0,
  pct_out_max = 100
)
```

Arguments

<code>res</code>	Long-form results table from <code>wilcoxauc()</code> .
<code>n</code>	Number of top markers to return per group. Default 10.
<code>auc_min</code>	Drop features with <code>auc < auc_min</code> . Default 0 (no filter); set to 0.5 to keep only features that are positive markers (more highly expressed in-group than out).
<code>pval_max</code>	Drop features with raw <code>pval > pval_max</code> . Default 1.
<code>padj_max</code>	Drop features with adjusted <code>padj > padj_max</code> . Default 1.
<code>pct_in_min</code>	Minimum percent (0-100) of in-group observations with non-zero feature value. Default 0.
<code>pct_out_max</code>	Maximum percent (0-100) of out-of-group observations with non-zero feature value. Default 100.

Value

tibble in wide form: a rank column (1..*n*) and one column per group containing the feature name of the top-ranked marker at that rank. Cells are NA for groups with fewer than *n* features that pass the filters.

See Also

`wilcoxauc()`, `top_markers_dds()`

Examples

```
set.seed(42)
exprs <- matrix(rpois(25 * 150, lambda = 2), nrow = 25,
  dimnames = list(paste0("G", 1:25), NULL))
y <- rep(c("A", "B", "C"), each = 50)

res <- wilcoxauc(exprs, y)

## top 10 markers per group, restricted to nominally significant,
## up-regulated features (auc > 0.5 means in-group > out-of-group).
top_markers(res, n = 10, auc_min = 0.5, pval_max = 0.05)
```

top_markers_dds	<i>Top markers per group from pseudobulk DESeq2 results</i>
-----------------	---

Description

Filters and ranks the long-form output of `pseudobulk_deseq2()` to give the most distinguishing features per group. The filtering arguments combine multiplicatively, then the top *n* features per group are kept by descending Wald statistic and pivoted into wide form. Counterpart to `top_markers()` for Wilcoxon-based results.

Usage

```
top_markers_dds(res, n = 10, pval_max = 1, padj_max = 1, lfc_min = 1)
```

Arguments

<code>res</code>	Long-form DESeq2 results from <code>pseudobulk_deseq2()</code> . Must have columns <code>group</code> , <code>feature</code> , <code>pvalue</code> , <code>padj</code> , <code>log2FoldChange</code> , and <code>stat</code> .
<code>n</code>	Number of top features to return per group. Default 10.
<code>pval_max</code>	Filter features with raw pvalue > <code>pval_max</code> . Default 1 (no filter).
<code>padj_max</code>	Filter features with adjusted <code>padj</code> > <code>padj_max</code> . Default 1 (no filter).
<code>lfc_min</code>	Filter features with <code>log2FoldChange</code> < <code>lfc_min</code> . Default 1 keeps only upregulated features; set to 0 to include downregulated as well, or -Inf to disable.

Value

tibble in wide form: a rank column (1..*n*) and one column per group containing the gene identifier of the top-ranked feature at that rank. Cells are NA for groups that have fewer than *n* features passing the filters.

See Also

`pseudobulk_deseq2()`, `top_markers()`

toy_sce	<i>Toy SingleCellExperiment object for examples and tests</i>
---------	---

Description

Builds a small deterministic SingleCellExperiment (20 genes x 300 cells, two cell types in the cell_type colData column) with the installed SingleCellExperiment version, so no serialized object needs to ship with the package. The same simulated counts as `toy_seurat()` are stored in the counts assay, with log-normalized values in logcounts. The generator restores the caller's random-number state.

Usage

```
toy_sce()
```

Details

Requires the suggested package SingleCellExperiment (Bioconductor).

Value

A SingleCellExperiment with counts and logcounts assays and a cell_type colData column with values "jurkat" and "t293".

See Also

`toy_seurat()`, `wilcoxauc()`

Examples

```
if (requireNamespace("SingleCellExperiment", quietly = TRUE)) {
  object_sce <- toy_sce()
  head(wilcoxauc(object_sce, "cell_type"))
}
```

toy_seurat	<i>Toy Seurat object for examples and tests</i>
------------	---

Description

Builds a small deterministic Seurat object (20 genes x 300 cells, two cell types in the cell_type metadata column) with the installed Seurat version, so no serialized object needs to ship with the package or be updated when Seurat changes its internal class structure. Counts are Poisson-simulated with a few upregulated marker genes per cell type, and log-normalized into the data layer. The generator restores the caller's random-number state, so calling it does not perturb reproducibility.

Usage

```
toy_seurat()
```

Details

Requires the suggested package Seurat.

Value

A Seurat object with counts and data layers and a cell_type metadata column with values "jurkat" and "t293".

See Also

[toy_sce\(\)](#), [wilcoxauc\(\)](#)

Examples

```
if (requireNamespace("Seurat", quietly = TRUE)) {
  object_seurat <- toy_seurat()
  head(wilcoxauc(object_seurat, "cell_type"))
}
```

wilcoxauc

Fast Wilcoxon rank-sum test and auROC across groups

Description

For every (feature, group) pair, computes the Wilcoxon rank-sum statistic comparing observations in that group against all other observations, and the area under the ROC curve as a measure of separability. P-values come from the standard Gaussian approximation to the U statistic with a tie correction. Returns one row per (feature, group) with effect-size and percent-expressed columns alongside the test statistics.

Usage

```
wilcoxauc(X, ...)

## S3 method for class 'seurat'
wilcoxauc(X, ...)

## S3 method for class 'Seurat'
wilcoxauc(
  X,
  group_by = NULL,
  assay = "data",
  groups_use = NULL,
```

```

    seurat_assay = "RNA",
    ...
)

## S3 method for class 'SingleCellExperiment'
wilcoxauc(X, group_by = NULL, assay = NULL, groups_use = NULL, ...)

## Default S3 method:
wilcoxauc(
  X,
  y,
  groups_use = NULL,
  verbose = TRUE,
  nthreads = 1,
  transposed = FALSE,
  ...
)

```

Arguments

X	Input data. One of: • a numeric feature-by-observation matrix or <code>data.frame</code>, • a sparse <code>dgCMatrix</code> of the same shape, • a disk-backed <code>DelayedMatrix</code> (e.g. HDF5-backed, from the <code>DelayedArray</code> / <code>HDF5Array</code> packages), which is processed in feature blocks so the whole matrix never has to be loaded in memory, • any other matrix-like class with an <code>as(., "dgCMatrix")</code> coercion method (e.g. <code>BPCells</code>), which is converted up front, • a Seurat (v3+) object, • a <code>SingleCellExperiment</code> object. X must not contain NA values. Unlike <code>stats::wilcox.test()</code>, which drops missing values per observation, <code>wilcoxauc()</code> errors on NA input rather than returning silently incorrect results; remove or impute missing values first.
...	Passed to the input-specific method.
group_by	For Seurat and <code>SingleCellExperiment</code> input, name of the metadata column that holds the group labels (e.g. "cluster"). For Seurat, defaults to <code>Idents(X)</code> .
assay	For Seurat, the layer name within the selected assay (e.g. "data", "counts", "scale.data"). For <code>SingleCellExperiment</code> , the assay name (e.g. "logcounts", "counts"). Defaults pick a sensible value per input class.
groups_use	Optional character vector restricting the test to a subset of groups in y (or group_by). Default NULL tests every group.
seurat_assay	For Seurat input, the name of the assay to pull from (e.g. "RNA"). Default "RNA".
y	For matrix input, a character/factor vector of group labels with length equal to <code>ncol(X)</code> . Ignored for Seurat / <code>SingleCellExperiment</code> input (use group_by instead).

verbose	Logical. Print warnings and informational messages. Default TRUE.
nthreads	Number of threads for the per-feature ranking of sparse (dgCMatrx) input. Default 1 (serial). Values above 1 split the ranking across threads; the result is identical regardless of the thread count. Only sparse input is parallelized – dense matrix and data.frame input are always processed serially. When running under R CMD check or on CRAN, keep this at the default so no more than two cores are used.
transposed	Set to TRUE when your observations (cells, samples) are in the rows of X and the features in the columns – i.e. X is the transpose of the default features-by-observations layout. The test then runs directly on that layout without materializing a transposed copy, which saves time and memory on large matrices. Same convention as the transposed argument of scater's calculatePCA() and calculateUMAP(). Only applies to matrix-like input (the Seurat / SingleCellExperiment dispatchers always extract features-by-observations). Default FALSE.

Details

Designed to be fast enough to run on whole-genome $\times$ hundred-thousand- cell single-cell matrices in seconds. Sparse dgCMatrx inputs are processed without densification. Convenience dispatchers extract the counts matrix and group labels from Seurat and SingleCellExperiment objects. See the getting-started vignette for an end-to-end example on a real dataset.

Value

table with the following columns:

- **feature** - feature name (e.g. gene name).
- **group** - group name.
- **avgExpr** - mean value of feature in group.
- **logFC** - difference of mean feature values between observations in the group vs out of the group. When the input is log-transformed expression (e.g. Seurat's "data" layer or logcounts), this difference of means is a log fold change. On raw (untransformed) values it is a plain difference of means, not a fold change.
- **statistic** - Wilcoxon rank sum U statistic.
- **auc** - area under the receiver operator curve.
- **pval** - nominal p value.
- **padj** - Benjamini-Hochberg adjusted p value.
- **pct_in** - Percent of observations in the group with non-zero feature value.
- **pct_out** - Percent of observations out of the group with non-zero feature value.

See Also

[top_markers\(\)](#) to summarize markers per group; [pseudobulk_deseq2\(\)](#) for a count-based pseudobulk alternative.

Examples

```
## generate a tiny toy dataset
set.seed(42)
exprs <- matrix(rpois(25 * 150, lambda = 2), nrow = 25,
               dimnames = list(paste0("G", 1:25), NULL))
y <- rep(c("A", "B", "C"), each = 50)

## on a dense matrix
head(wilcoxauc(exprs, y))

## restrict the comparison to a subset of groups
head(wilcoxauc(exprs, y, c('A', 'B'))))

## on a sparse matrix
exprs_sparse <- as(exprs, 'dgCMatrix')
head(wilcoxauc(exprs_sparse, y))

## on a Seurat object (>= v3)
if (requireNamespace("Seurat", quietly = TRUE) &&
    packageVersion("Seurat") >= "3.0") {
  object_seurat <- toy_seurat()
  head(wilcoxauc(object_seurat, 'cell_type'))
}

## on a SingleCellExperiment object
if (requireNamespace("SingleCellExperiment", quietly = TRUE)) {
  object_sce <- toy_sce()
  head(wilcoxauc(object_sce, 'cell_type'))
}
```

Index

`collapse_counts`, 2
`collapse_counts()`, 4–6, 12
`compute_hash`, 4
`compute_hash()`, 3

`nnzeroGroups`, 4
`nnzeroGroups()`, 12

`pseudobulk_deseq2`, 5
`pseudobulk_deseq2()`, 3, 8–11, 13, 15, 19
`pseudobulk_one_vs_all`, 7
`pseudobulk_one_vs_all()`, 5, 10, 11
`pseudobulk_pairwise`, 8
`pseudobulk_pairwise()`, 5, 8, 11, 13
`pseudobulk_within`, 10
`pseudobulk_within()`, 5, 8, 10

`rank_matrix`, 11

`stats::wilcox.test()`, 18
`sumGroups`, 12
`sumGroups()`, 5
`summarize_dge_pairs`, 13
`summarize_dge_pairs()`, 5, 6, 9, 10

`top_markers`, 14
`top_markers()`, 15, 19
`top_markers_dds`, 15
`top_markers_dds()`, 6, 8, 14
`toy_sce`, 16
`toy_sce()`, 17
`toy_seurat`, 16
`toy_seurat()`, 16

`wilcoxauc`, 17
`wilcoxauc()`, 4, 5, 11, 12, 14, 16, 17