Package ‘CellaRepertorium’

March 20, 2024

Type Package

Title Data structures, clustering and testing for single cell immune receptor repertoires (scRNAseq RepSeq/AIRR-seq)

Version 1.12.0

Description Methods to cluster and analyze high-throughput single cell immune cell repertoires, especially from the 10X Genomics VDJ solution. Contains an R interface to CD-HIT (Li and Godzik 2006). Methods to visualize and analyze paired heavy-light chain data. Tests for specific expansion, as well as omnibus oligoclonality under hypergeometric models.

License GPL-3

Depends R (>= 4.0)

Imports dplyr, tibble, stringr, Biocstrings, Rcpp, reshape2, methods, rlang (>= 0.3), purrr, Matrix, S4Vectors, BiocGenerics, tidyr, forcats, progress, stats, utils, generics, glue

Suggests testthat, readr, knitr, rmarkdown, ggplot2, BiocStyle, ggdendro, broom, lme4, RColorBrewer, SingleCellExperiment, scater, broom.mixed, cowplot, igraph, ggraph

LinkingTo Rcpp

VignetteBuilder knitr

Encoding UTF-8

NeedsCompilation yes

RoxygenNote 7.1.2

URL https://github.com/amcdavid/CellaRepertorium

BugReports https://github.com/amcdavid/CellaRepertorium/issues

Roxygen list(markdown = TRUE)

biocViews RNASeq, Transcriptomics, SingleCell, TargetedResequencing, Technology, ImmunoOncology, Clustering
Collate 'AllClasses.R' 'ContigCellDB-methods.R' 'RcppExports.R'
   'auxiliary.R' 'ccdb_join.R' 'cdhit-methods.R'
   'cluster-testing.R' 'clustering-methods.R' 'data.R'
   'ggplot2-utils.R' 'pairing-methods.R' 'reexports.R'
   'permutation-testing.R' 'plot_clustering.R' 'plotting.R'
   'pseudo-bulk.R' 'utility.R' 'vdjcellranger-methods.R'

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

git_branch RELEASE_3_18

git_last_commit eccffbf

git_last_commit_date 2023-10-24

Repository Bioconductor 3.18

Date/Publication 2024-03-20

Author Andrew McDavid [aut, cre],
   Yu Gu [aut],
   Erik VonKaenel [aut],
   Aaron Wagner [aut],
   Thomas Lin Pedersen [ctb]

Maintainer Andrew McDavid <Andrew_McDavid@urmc.rochester.edu>

R topics documented:

.cluster_permute_test ........................................ 3
canonicalize_cell ........................................... 4
canonicalize_cluster ......................................... 5
ccdb_ex ....................................................... 7
ccdb_join ..................................................... 7
cdhit ......................................................... 8
cdhit_ccdb .................................................. 9
cland ......................................................... 10
cluster_filterset ............................................ 11
cluster_germline ............................................. 12
cluster_permute_test ......................................... 12
cluster_plot ................................................ 14
cluster_test_by ............................................. 15
ContigCellDB .................................................. 16
contigs_qc .................................................. 18
crosstab_by_celltype ....................................... 19
cross_tab_tbl ................................................ 19
entropy ...................................................... 20
equalize_ccdb ............................................... 21
fancy_name_contigs ......................................... 22
filter_cdb .................................................. 23
fine_clustering .............................................. 24
fine_cluster_seqs .......................................... 25
generate_pseudobulk ....................................... 26
.cluster_permute_test

Description

Cell permutation tests (internal)

Usage

`.cluster_permute_test(
  labels,
  covariates,
  strata,
  statistic,
  contrasts,
  n_perm,
  alternative,
  ...
)

Arguments

labels factor of length n
covariates data.frame of length n
strata factor
statistic function of label (vector) and covariate (data.frame). If this returns a vector, then by default each level will be compared against each other, pairwise, but see the next section.
canonicalize_cell

contrasts an optional list of numeric vectors. Each will be dotted with the statistic, or optionally a matrix provided in which case each row would be tested one-by-one.
n_perm number of permutations to run
alternative character naming the direction statistic should be fall under the alternative hypothesis
... passed along to statistic

Value

a list containing the observed value of the statistic, the permuted values of the statistic, its expectation (under independence), a p-value, and the Monte Carlo standard error (of the expected value).

---

canonicalize_cell | Find a canonical contig to represent a cell

Description

Using filtering in contig_filter_args and sorting in tie_break_keys and order find a single, canonical contig to represent each cell Fields in contig_fields will be copied over to the cell_tbl.

Usage

canonicalize_cell(
  ccdb,
  contig_filter_args = TRUE,
  tie_break_keys = c("umis", "reads"),
  contig_fields = tie_break_keys,
  order = 1,
  overwrite = TRUE
)

Arguments

ccdb ContigCellDB()
contig_filter_args
an expression passed to dplyr::filter(). Unlike filter, multiple criteria must be & together, rather than using commas to separate. These act on ccdb$contig_tbl
tie_break_keys (optional) character naming fields in contig_tbl that are used sort the contig table in descending order. Used to break ties if contig_filter_args does not return a unique contig for each cluster
contig_fields Optional fields from contig_tbl that will be copied into the cluster_tbl from the canonical contig.
### canonicalize_cluster

Find a canonical contig to represent a cluster

**Description**

Find a canonical contig to represent a cluster

**Usage**

```r
canonicalize_cluster(
  ccdb,
  contig_filter_args,
  tie_break_keys = character(),
  order = 1,
  representative = ccdb$cluster_pk[1],
)```

---

**order**

The rank order of the contig, based on `tie_break_keys` to return. If `tie_break_keys` included an ordered factor (such as `chain`) this could be used to return the second chain.

**overwrite**

logical – should non-key fields in `y` be overwritten using `x`, or should a suffix (".y") be added

**Value**

`ContigCellDB()` with some number of clusters/contigs/cells but with "canonical" values copied into `cell_tbl`

**See Also**

`canonicalize_cluster()`

**Examples**

```r
# Report beta chain with highest umi-count, breaking ties with reads
data(ccdb_ex)
beta = canonicalize_cell(ccdb_ex, chain == 'TRB',
  tie_break_keys = c('umis', 'reads'),
  contig_fields = c('umis', 'reads', 'chain', 'v_gene', 'd_gene', 'j_gene'))
head(beta$cell_tbl)

# Stable: only adds fields to `cell_tbl`
stopifnot(dplyr::all_equal(beta$cell_tbl[ccdb_ex$cell_pk],
  ccdb_ex$cell_tbl[ccdb_ex$cell_pk], ignore_row_order = TRUE))

# Report cdr3 with highest UMI count, but only when > 5 UMIs support it
umi5 = canonicalize_cell(ccdb_ex, umis > 5,
  tie_break_keys = c('umis', 'reads'), contig_fields = c('umis', 'cdr3'))
stopifnot(all(umi5$cell_tbl$umis > 5, na.rm = TRUE))
```
canonicalize_cluster

```r
contig_fields = c("cdr3", "cdr3_nt", "chain", "v_gene", "d_gene", "j_gene"),
overwrite = TRUE
)
```

Arguments

- **ccdb**: `ContigCellDB()`
- **contig_filter_args**: an expression passed to `dplyr::filter()`. Unlike `filter`, multiple criteria must be & together, rather than using commas to separate. These act on `ccdb$contig_tbl`
- **tie_break_keys** (optional): character naming fields in `contig_tbl` that are used sort the contig table in descending order. Used to break ties if `contig_filter_args` does not return a unique contig for each cluster
- **order**: The rank order of the contig, based on `tie_break_keys` to return. If `tie_break_keys` included an ordered factor (such as chain) this could be used to return the second chain.
- **representative**: an optional field from `contig_tbl` that will be made unique. Serve as a surrogate cluster_pk.
- **contig_fields**: Optional fields from `contig_tbl` that will be copied into the `cluster_tbl` from the canonical contig.
- **overwrite**: logical – should non-key fields in `y` be overwritten using `x`, or should a suffix (".y") be added

Value

`ContigCellDB()` with some number of clusters/contigs/cells but with "canonical" values copied into `cluster_tbl`

See Also

- `canonicalize_cell()`, `left_join_warn()`

Examples

```r
library(dplyr)
data(ccdb_ex)
ccdb_ex_small = ccdb_ex
ccdb_ex_small$cell_tbl = ccdb_ex_small$cell_tbl[1:200,]
ccdb_ex_small = cdhit_ccdb(ccdb_ex_small,
sequence_key = 'cdr3_nt', type = 'DNA', cluster_name = 'DNA97',
identity = .965, min_length = 12, G = 1)
ccdb_ex_small = fine_clustering(ccdb_ex_small, sequence_key = 'cdr3_nt', type = 'DNA')

# Canonicalize with the medoid contig is probably what is most common
ccdb_medoid = canonicalize_cluster(ccdb_ex_small)

# But there are other possibilities.
# To pass multiple "AND" filter arguments must use &
A preconstructed ContigClusterDB from the contigs_qc data

Usage
data(ccdb_ex)

Format
ccdb_ex = ContigCellDB_10XVDJ(contigs_qc, contig_pk = c('pop', 'sample', 'barcode', 'contig_id'), cell_pk = c('pop', 'sample', 'barcode'))

See Also
contigs_qc

Join dataframe or SingleCellExperiment object with ContigCellDB object

Usage
ccdb_join(template, ccdb, join_fun = dplyr::left_join, by = ccdb$cell_pk)

Arguments
template data.frame or SingleCellExperiment object to be joined with ccdb.
ccdb A ContigCellDB object.
join_fun Function used for the join operation.
by A character vector of variables to join by.
Value

ContigCellDB()

Examples

data(ccdb_ex)
to_join = dplyr::bind_rows(ccdb_ex$cell_tbl[1:10,],
dplyr::tibble(barcode = c('extra1', 'extra2'), sample = LETTERS[1:2],
pop = LETTERS[1:2]))
ccdb_join(to_join, ccdb_ex)

Description

CDHIT is a greedy algorithm to cluster amino acid or DNA sequences based on a minimum identity. By default, in this package it is configured perform ungapped, global alignments with no clipping at start or end. The identity is the number of identical characters in alignment divided by the full length of the shorter sequence. Set s < 1 to change the minimum coverage of the shorter sequence, which will allow clipping at start or end. Changing G = 0 changes the meaning of the identity to be the number of identical characters in the alignment divided by the length of the alignment. In this case, you must also set the alignment coverage controls aL, AL, aS, AS.

Usage

cdhit(
  seqs,
  identity = NULL,
  kmerSize = NULL,
  min_length = 6,
  s = 1,
  G = 1,
  only_index = FALSE,
  showProgress = interactive(),
  ...
)

Arguments

seqs      AAseq or DNAseq
identity  minimum proportion identity
kmerSize  word size. If NULL, it will be chosen automatically based on the identity. You may need to lower it below 5 for AAseq with identity less than .7.
min_length  Minimum length for sequences to be clustered. An error if something smaller is passed.
Use `cdhit()` to cluster a `ContigCellDB()`

**Description**

See [https://github.com/weizhongli/cdhit/wiki/3.-User’s-Guide#CDHIT](https://github.com/weizhongli/cdhit/wiki/3.-User’s-Guide#CDHIT) for details on other potential arguments to `...`. These will override any default values.

**Usage**

```r
cdhit_ccdb(
  ccdb,
  sequence_key,
  type = c("DNA", "AA"),
  cluster_pk = "cluster_idx",
  ...)
```
Arguments

ccdb An object of class ContigCellDB()
description character naming the column in the contig_tbl containing the sequence to be clustered
type one of 'DNA' or 'AA'
cluster_pk character specifying key, and name for the clustering.
	Arguments passed on to cdhit
	identity minimum proportion identity
	kmerSize word size. If NULL, it will be chosen automatically based on the identity. You may need to lower it below 5 for AAseq with identity less than .7.
	min_length Minimum length for sequences to be clustered. An error if something smaller is passed.
s fraction of shorter sequence covered by alignment.
showProgress show a status bar
G 1 for global alignment, 0 for local. If doubt, pick global.

Value

ContigCellDB()

See Also

cdhit()

Examples

data(ccdb_ex)
res = cdhit_ccdb(ccdb_ex, 'cdr3_nt', type = 'DNA', cluster_name = 'DNA97', identity = .965, min_length = 12, G = 1)
res$cluster_tbl
res$contig_tbl
res$cluster_pk

cland

Cluster "And" intersection

Description

For each contig present in both X and Y, a new cluster is defined that combines cluster identities in both X and Y. In the resulting ContigCellDB, two contigs are in the same cluster if they are in the same cluster in X and the same cluster in Y. X and Y must have matching contig_pk. The contig_tbl has fields from X for contigs present in both X and Y. The cell_tbl from X is carried forward unchanged, while the cluster_tbl in the result contains the mapping between the ancestral clustering, and the derived.
Usage

cland(X, Y, new_pk)

Arguments

X ContigCellDB
Y ContigCellDB
new_pk optional character naming the new pk.

Examples

data(ccdb_ex)
ccdb_germ = cluster_germline(ccdb_ex, cluster_pk = 'germline_idx')
ccdb_cdr3 = cdhit_ccdb(ccdb_ex, 'cdr3_nt', type = 'DNA',
cluster_name = 'DNA97', identity = .965, min_length = 12, G = 1)
ccdb_cdr3 = cland(ccdb_cdr3, ccdb_germ)

cluster_filterset  A filtration of clusters

Description

Return clusters that match all provided conditions

Usage

cluster_filterset(min_number = 0, min_freq = 0, white_list = NULL)

Arguments

min_number integer At least this many cells
min_freq numeric At least this frequency
white_list data.frame keyed by cluster_pk that must match

Value

object representing the filtration (currently a list)

Examples

cluster_filterset(min_number = 1, min_freq = 0)
cluster_permute_test

Tests for independence between labels and covariates using permutation of cells

Description

This tests a statistic for association between labels (for instance, cluster/clonal ID) and covariates (for instance, subject or treatment) by permuting the link between the two. Each observation represents a cell. statistic is any function of labels
Usage

cluster_permute_test(
  ccdb,
  cell_covariate_keys,
  cell_label_key = ccdb$cluster_pk,
  cell_stratify_keys,
  statistic,
  contrasts = NULL,
  n_perm,
  alternative = c("two.sided", "less", "greater"),
  sanity_check_strata = TRUE,
  ...
)

Arguments

ccdb ContigCellDB

cell_covariate_keys character naming fields in ccdb$cell_tbl

cell_label_key character naming a single field in ccdb$cell_tbl

cell_stratify_keys optional character naming fields in ccdb$cell_tbl under which permutations of cell_label_key will occur. This means that the test will occur conditional on these covariates. Must be disjoint from cell_covariate_keys.

statistic function of label (vector) and covariate (data.frame). If this returns a vector, then by default each level will be compared against each other, pairwise, but see the next section.

contrasts an optional list of numeric vectors. Each will be dotted with the statistic, or optionally a matrix provided in which case each row would be tested one-by-one.

n_perm number of permutations to run

alternative character naming the direction statistic should be fall under the alternative hypothesis

sanity_check_strata logical, should cell_stratify_keys be checked for sanity?

... passed to statistic

Value

a list containing the observed value of the statistic, the permuted values of the statistic, its expectation (under independence), a p-value, and the Monte Carlo standard error (of the expected value).

See Also

purity()
Examples

```r
library(dplyr)
# covariate should name one or more columns in `cell_tbl`

cluster_idx = c(1, 1, 2, 2, 3, 3)
subject = c('A', 'A', 'B', 'B', 'C', 'C')
contig_tbl = tibble(contig_pk = seq_along(cluster_idx), cluster_idx, subject)
ccdb_test = ContigCellDB(contig_tbl = contig_tbl, contig_pk = 'contig_pk',
cell_pk = c('contig_pk', 'subject', 'cluster_idx'), cluster_pk = 'cluster_idx')
ccdb_test$cell_tbl

clust_test = cluster_permute_test(ccdb_test, 'subject', 'cluster_idx',
statistic = purity, n_perm = 50)
library(ggplot2)
plot_permute_test(perm_test = clust_test)
tidy.PermuteTest(clust_test)
```

cluster.plot

Make a plot showing properties of the clustering

Description

The number of elements per cluster and the average distance between the medoid and other elements are plotted.

Usage

```r
cluster.plot(cdb, return_plotlist = FALSE)
```

Arguments

- **cdb**: A fine_clustering ContigCellDB object
- **return_plotlist**: should a list of ggplot2 plots be returned. If FALSE, a cowplot composite is returned.

Value

- a cowplot composite or a list of plots.

Examples

```r
library(dplyr)
data(ccdb_ex)
ccdb_ex_small = ccdb_ex
ccdb_ex_small$cell_tbl = ccdb_ex_small$cell_tbl[1:200,]
ccdb_ex_small = cdhit_ccdb(ccdb_ex_small, sequence_key = 'cdr3_nt', type = 'DNA', cluster_name = 'DNA97', identity = .965, min_length = 12, G = 1)
```
```r
ccdb_ex_small = fine_clustering(ccdb_ex_small, sequence_key = 'cdr3_nt', type = 'DNA')

# Canonicalize with the medoid contig is probably what is most common
ccdb_medoid = canonicalize_cluster(ccdb_ex_small)

# But there are other possibilities.
# To pass multiple "AND" filter arguments must use &
ccdb_umi = canonicalize_cluster(ccdb_ex_small, contig_filter_args = chain == 'TRA' & length > 500, tie_break_keys = 'umis', contig_fields = c('chain', 'length'))
ccdb_umi$cluster_tbl %>% dplyr::select(chain, length) %>% summary()
cluster_plot(ccdb_ex_small)
```

---

**cluster_test_by**

**Test clusters for differential usage**

**Description**

Typically one will want to stratify by chain by calling `cluster_test_by`, as this will calculate the number of cell "trials" separately depending on the chain recovered.

**Usage**

```r
cluster_test_by(ccdb, fields = "chain", tbl = "cluster_tbl", ...)
```

```r
cluster_logistic_test(
  formula,  # the right-hand side of a glmer or glm-style formula.
  ccdb,  
  filterset = cluster_filterset(),  
  contig_filter_args = TRUE,  
  tie_break_keys = c("umis", "reads"),  
  add_cluster_tbl = FALSE,
  keep_fit = FALSE,
  fitter = glm_glmer,
  silent = FALSE
)
```

**Arguments**

- **ccdb** | ContigCellDB()
- **fields** | character naming fields in tbl
- **tbl** | one of contig_tbl, cell_tbl or cluster_tbl
- **...** | passed to `cluster_logistic_test`
- **formula** | the right-hand side of a glmer or glm-style formula.
- **filterset** | a call to `cluster_filterset()` that will be used to subset clusters.
contig_filter_args
- an expression passed to `dplyr::filter()`. Unlike filter, multiple criteria must be & together, rather than using commas to separate. These act on `ccdb$contig_tbl`

tie_break_keys
- (optional) character naming fields in `contig_tbl` that are used sort the contig table in descending order. Used to break ties if `contig_filter_args` does not return a unique contig for each cluster

add_cluster_tbl
- logical should the output be joined to the `cluster_tbl`?

keep_fit
- logical as to whether the fit objects should be returned as a list column

fitter
- a function taking arguments `formula`, `data`, `is_mixed` and `keep_fit` that is run on each cluster. Should return a `tibble` or `data.frame`

silent
- logical. Should warnings from fitting functions should be suppressed?

**Value**
- table with one row per cluster/term.

**Functions**
- `cluster_test_by`: split ccdb and conduct tests within strata

**Examples**
```r
library(dplyr)
data(ccdb_ex)
ccdb_ex = cluster_germline(ccdb_ex)
trav1 = filter(ccdb_ex$cluster_tbl, v_gene == 'TRAV1')
cluster_logistic_test(~pop + (1|sample), ccdb_ex,
  filterset = cluster_filterset(white_list= trav1))
# Fixed effect analysis of each cluster, by chain
prev4 = ccdb_ex$contig_tbl %>% group_by(cluster_idx) %>%
  summarize(n()) %>% filter(n()>= 4)
cluster_test_by(ccdb = ccdb_ex, fields = 'chain',
  tbl = 'cluster_tbl', formula = ~ pop, filterset = cluster_filterset(white_list= prev4))
```

---

**ContigCellDB**

**Construct a ContigCellDB**

**Description**
- Construct a ContigCellDB
ContigCellDB

Usage

ContigCellDB(
  contig_tbl,
  contig_pk,
  cell_tbl,
  cell_pk,
  cluster_tbl,
  cluster_pk = character(),
  equalize = TRUE
)

ContigCellDB_10XVDJ(
  contig_tbl,
  contig_pk = c("barcode", "contig_id"),
  cell_pk = "barcode",
  ...
)

Arguments

contig_tbl a data frame of contigs, and additional fields describing their properties
contig_pk character vector naming fields in contig_tbl that uniquely identify a row/contig
cell_tbl a data frame of cell barcodes, and (optional) additional fields describing their properties
cell_pk character vector naming fields in cell_tbl that uniquely identify a cell barcode
cluster_tbl A data frame that provide cluster assignments for each contig
cluster_pk If cluster_tbl was provided, a character vector naming fields in cluster_tbl that uniquely identify a cluster
equalize logical. Should the contig, cells and clusters be equalized by taking the intersection of their common keys?
...
... passed to ContigCellDB()

Value

ContigCellDB

Functions

- ContigCellDB_10XVDJ: provide defaults that correspond to identifiers in 10X VDJ data

Accessors/mutators

See $.ContigCellDB-method for more on how to access and mutate slots. See mutate_cdb() and filter_cdb() for endomorphic filtering/mutation methods See split_cdb() to split into a list, and rbind.ContigCellDB() for the inverse operation.
contigs_qc

Filtered and annotated contigs of TCR from mice

Description

Data for c57bl6 and balbc mice TCR were downloaded from 10x Genomics website as shown in `system.file('script/10XMouseTCR_v3_chem.R', package = 'CellaRepertorium')`. Additional processing of these data is done in the vignette `mouse_tcell_qc` and are serialized to serve as examples for other vignettes and documentation.

Usage

`data(contigs_qc)`

Format

A data frame of 3399 contigs and 22 fields, all except 4 are originally defined in https://support.10xgenomics.com/single-cell-vdj/software/pipelines/latest/output/annotation#contig. The following fields were defined ex post facto.

1. anno_file: Path to original csv file
2. pop: Mouse strain.
3. sample: An artificial "replicate" from the original data defined by subsampling with replacement
4. celltype: The putative cell type of the contig.

---

**crosstab_by_celltype**  
*Count contig UMIs by celltype*

**Description**
Count contig UMIs by celltype

**Usage**
crosstab_by_celltype(ccdb)

**Arguments**
ccdb  
A ContigCellDB object

**Value**
a table, keyed by cell_pk counting UMIs per celltype

**See Also**
guess_celltype()

**Examples**
data(ccdb_ex)
nrow(ccdb_ex$cell_tbl)
total_umi = crosstab_by_celltype(ccdb_ex)
nrow(total_umi)

---

**cross_tab_tbl**  
*Generate a 2d cross tab using arbitrary numbers of columns as factors*

**Description**
As many rows as unique combs of x_fields  
As many columns as unique combs of y_fields  
No NA.

**Usage**
cross_tab_tbl(tbl, x_fields, y_fields)
entropy

**Arguments**

- `tbl`  
  data.frame
- `x_fields`  
  character fields in tbl
- `y_fields`  
  character fields in tbl

**Value**

- tibble

**Examples**

```r
cross_tab_tbl(mtcars, c('cyl', 'gear'), 'carb')
```

---

**entropy**  
*Calculate the entropy of a vector*

**Description**

Calculate the entropy of a vector

**Usage**

```r
entropy(v, pseudo_count = length(v)/1000, na.action = na.fail)
np(v, p = 0.05, pseudo_count = p/5, na.action = na.fail)
modal_category(v, na.action = na.fail)
```

**Arguments**

- `v`  
  categorical vector
- `pseudo_count`  
  number of pseudo counts to add on, to stabilize empty categories
- `na.action`  
  how to handle NA values
- `p`  
  proportion threshold

**Value**

- the sample entropy

**Functions**

- `np`: The number of categories exceeding `p` proportion of the total
- `modal_category`: The modal category of `v`. Ties are broken by lexicographic order of the factor levels.
**Examples**

\[
v_2 = \text{gl}(2, 4) \\
v_4 = \text{gl}(4, 4) \\
\text{stopifnot}(\text{entropy}(v_2) < \text{entropy}(v_4)) \\
v_{\text{empty}} = v_2[1:4] \# \text{empty level 2} \\
\text{stopifnot}(\text{is.finite}(\text{entropy}(v_{\text{empty})))) \# \text{pseudo\_count}
\]

\[
\text{np}(v_4, p = .2, \text{pseudo\_count} = 0) \\
\text{np}(v_4, p = .25, \text{pseudo\_count} = 0) \\
\text{np}(v_4, p = .25, \text{pseudo\_count} = .0001)
\]

\[
\text{modal\_category}(v_4) \\
\text{modal\_category}(v_4[-1])
\]

**equalize_ccdb**

*Take the intersection of keys in tables in x*

**Description**

The cells in `cell_tbl`, and clusters in `cluster_tbl` can potentially be a superset of the `contig_tbl`.

**Usage**

\[
\text{equalize\_ccdb}(x, \text{cell} = \text{TRUE}, \text{contig} = \text{TRUE}, \text{cluster} = \text{TRUE}, \text{sort} = \text{FALSE})
\]

**Arguments**

- `x`: `ContigCellDB()`
- `cell`: logical equalize cells
- `contig`: logical equalize contigs
- `cluster`: logical equalize clusters
- `sort`: logical should equalized fields also be `order()`ed by their primary keys?

**Details**

- `equalize\_ccdb(x, \text{cell} = \text{TRUE})` trims cells that aren’t in `contig_tbl` or `cluster_tbl`.
- `equalize\_ccdb(x, \text{cluster} = \text{TRUE})` trims clusters that aren’t in `contig_tbl`.
- `equalize\_ccdb(x, \text{contig} = \text{TRUE})` trims contigs that aren’t `cell_tbl` or `cluster_tbl`.

**Value**

`ContigCellDB()`

**Default equalization**

Modification to `contig_tbl` (with `$`) always equalizes contigs and clusters. Modification to `cell_tbl` equalizes only contigs. Modification to `cluster_tbl` equalizes contigs and clusters.
Examples

```r
library(dplyr)
tbl = tibble(clust_idx = gl(3, 2), cell_idx = rep(1:3, times = 2), contig_idx = 1:6)
ccdb = ContigCellDB(tbl, contig_pk = c('cell_idx', 'contig_idx'),
cell_pk = 'cell_idx', cluster_pk = 'clust_idx')
# 3 cells
ccdb
ccdb$cell_tbl = bind_rows(ccdb$cell_tbl, tibble(cell_idx = 0))
# 4 cells now
ccdb
# 3 cells again
equalize_ccdb(ccdb)
# remove all contigs from cell 1, and one contig from cell 2
ccdb$contig_tbl = ccdb$contig_tbl[-c(1, 2, 4),]
# no changes to cell_tbl yet
ccdb
# trim cell_tbl to 2 cells, keep all clusters
equalize_ccdb(ccdb, cluster = FALSE)
# trim both cells and clusters
equalize_ccdb(ccdb, cluster = TRUE)
```

---

**fancy_name_contigs**

Generate a legible name for a series of contigs

**Description**

Generate a legible name for a series of contigs

**Usage**

```r
fancy_name_contigs(contig_tbl, prefix)
```

**Arguments**

- `contig_tbl` : An all_contig_annotations.csv file, output from VDJ Cell ranger. Importantly, this should contain columns chain, v_gene, d_gene, j_gene
- `prefix` : an optional prefix added to each contig, eg, possibly a sample id.

**Value**

character

**Examples**

```r
library(dplyr)
contig_anno_path = system.file('extdata', 'all_contig_annotations_balbc_1.csv.xz',
package = 'CellaRepertorium')
contig_anno = readr::read_csv(contig_anno_path)
contig_anno = contig_anno %>% mutate(fancy_name =
```
fancy_name_contigs(. , prefix = 'b6_1')
stoptifnot(!any(duplicated(contig_anno$fancy_name)))

filter_cdb Create new or update existing columns of ContigCellDB tables

Description

Create new or update existing columns of ContigCellDB tables

Usage

filter_cdb(ccdb, ..., tbl = "contig_tbl")
mutate_cdb(ccdb, ..., tbl = "contig_tbl")

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ccdb</td>
<td>ContigCellDB()</td>
</tr>
<tr>
<td>...</td>
<td>name and value pair of column that will be updated</td>
</tr>
<tr>
<td>tbl</td>
<td>character. One of contig_tbl, cell_tbl or cluster_tbl, naming the table to be updated.</td>
</tr>
</tbody>
</table>

Value

ContigCellDB object with updated table

Functions

- filter_cdb: Filter rows of a table in a ContigCellDB object

See Also

dplyr::mutate()
dplyr::filter()

Examples

data(ccdb_ex)
subset_contig = filter_cdb(ccdb_ex, full_length, productive == 'True',
high_confidence, chain != 'Multi', nchar(cdr3) > 5)
subset_cell = filter_cdb(ccdb_ex, sample == 4, tbl = 'cell_tbl')
data(ccdb_ex)
new_contig = mutate_cdb(ccdb_ex, new_col = 1)
new_cell = mutate_cdb(ccdb_ex, new_col = 1, tbl = 'contig_tbl')
Perform additional clustering of sequences within groups

**Usage**

```r
fine_clustering(
  ccdb,
  sequence_key,
  type,
  max_affinity = NULL,
  keep_clustering_details = FALSE,
  ...)
```

**Arguments**

- `ccdb`: A `ContigCellDB()` object.
- `sequence_key`: character naming column in `contig_tbl` with sequence.
- `type`: 'AA' or 'DNA'.
- `max_affinity`: numeric naming the maximal affinity for the sparse affinity matrix that is constructed. Not currently used.
- `keep_clustering_details`: logical – should output of `fine_cluster_seqs` be kept as a list column.
- `...`: Arguments passed on to `fine_cluster_seqs`.

**Value**

A `ContigCellDB()` object with updated `contig_tbl` and `cluster_tbl`.

**Examples**

```r
library(dplyr)
data(ccdb_ex)
ccdb_ex_small = ccdb_ex
ccdb_ex_small$cell_tbl = ccdb_ex_small$cell_tbl[1:200,]
ccdb_ex_small = cdhit_ccdb(ccdb_ex_small,
  sequence_key = 'cdr3_nt',
  type = 'DNA',
  cluster_name = 'DNA97',
)
fine_cluster_seqs

Calculate distances and perform hierarchical clustering on a set of sequences

Description

The distances between AA sequences is defined to be 1-score/max(score) times the median length of the input sequences. The distances between nucleotide sequences is defined to be edit_distance/max(edit_distance) times the median length of input sequences.

Usage

fine_cluster_seqs(
  seqs,
  type = "AA",
  big_memory_brute = FALSE,
  method = "levenshtein",
  substitution_matrix = "BLOSUM100",
  cluster_fun = "none",
  cluster_method = "complete"
)

Arguments

seqs character vector, DNASTRINGSet or AASTRINGSet
type character either AA or DNA specifying type of seqs
big_memory_brute attempt to cluster more than 4000 sequences? Clustering is quadratic, so this will take a long time and might exhaust memory
method one of 'substitutionMatrix' or 'levenshtein'
substitution_matrix a character vector naming a substitution matrix available in Biostrings, or a substitution matrix itself
cluster_fun character, one of "hclust" or "none", determining if distance matrices should also be clustered with hclust
cluster_method character passed to hclust
generate_pseudobulk

Value
list

See Also
hclust(), Biostrings::stringDist()

Examples

```r
fasta_path = system.file('extdata', 'demo.fasta', package='CellaRepertorium')
aaseq = Biostrings::readAAStringSet(fasta_path)[1:100]
cls = fine_cluster_seqs(aaseq, cluster_fun = 'hclust')
plot(cls$cluster)
```

generate_pseudobulk  Generate "pseudobulk" data from a ContigCellDB

Description
Tabulate contigs with a unique combination of class_keys per total_keys. For instance, total_keys might be a sample identifier, and class_keys might be the V- and J- gene identities. The idea is that this might mimic the data generated in a bulk experiment.

Usage

```r
generate_pseudobulk(ccdb, class_keys, total_keys, type = c("cell", "umi"))
```

Arguments

- **ccdb** `ContigCellDB()`
- **class_keys** character naming fields in contig_tbl that define unique classes of the repertoire
- **total_keys** character naming fields to be conditioned upon when calculating the total.
- **type** one of "cell" or "umi"

Details
This function is currently rather 10x-specific, in that it is assumed that columns barcode and umis exist.

Value
tibble

Examples

```r
data(ccdb_ex)
ccdb_ex = cluster_germline(ccdb_ex)
pseudo = generate_pseudobulk(ccdb_ex, c('v_gene', 'j_gene', 'chain'), c('pop', 'sample'))
```
guess_celltype

Guess the cell type of a contig from the chain ID

Description
This function is likely dependent on annotations from 10X and may change or break as their pipeline changes.

Usage
guess_celltype(chain)

Arguments
chain character which will be parsed to try to infer celltype

Value
contig table with celltype column

See Also
crosstab_by_celltype()

Examples
data(ccdb_ex)
table(guess_celltype(ccdb_ex$contig_tbl$chain))

hushWarning

Selectively muffle warnings based on output

Description
Selectively muffle warnings based on output

Usage
hushWarning(expr, regexp)

Arguments
expr an expression
regexp a regexp to be matched (with str_detect)
**Value**

the result of expr

**Examples**

```r
CellaRepertorium:::hushWarning(warning('Beware the rabbit'), 'rabbit')
CellaRepertorium:::hushWarning(warning('Beware the rabbit'), 'hedgehog')
```

---

**ig_chain_recode**

Categorize the pairing present in a cell

**Description**

For each cell (defined by `ccdb$cell_pk`) count the number of each level of `chain_key` occurs, and cross tabulate. Also for each cell, paste together all values `chain_key`. Return a tibble, keyed by cells that includes the counts of the chains, the `raw_chain_type` and any additional output from running `chain_recode_fun`.

**Usage**

```r
ig_chain_recode(tbl)
tcr_chain_recode(tbl)
enumerate_pairing(ccdb, chain_key = "chain", chain_recode_fun = NULL)
```

**Arguments**

- `tbl` output from `enumerate_pairing` containing TRA/TRB or IGH/IHK/IHL columns
- `ccdb` ContigCellDB
- `chain_key` character naming the field in the `contig_tbl` identifying chain
- `chain_recode_fun` a function that operates on the output of this function that further reduces the chain combinations to some other summary. Set to 'guess' to apply functions that may work for 10X data or NULL to skip. See `CellaRepertorium:::tcr_chain_recode` for an example.

**Value**

a tibble keyed by cells.

**Functions**

- `ig_chain_recode`: Recode a table with IG chains
- `tcr_chain_recode`: Recode a table with TCR chains
map_axis_labels

Examples

data(ccdb_ex)
enumerate_pairing(ccdb_ex)
enumerate_pairing(ccdb_ex, chain_recode_fun = 'guess')

---

map_axis_labels  Color axis labels

Description

Color axis labels

Usage

map_axis_labels(
  plt,
  label_data_x = NULL,
  label_data_y = NULL,
  aes_label,
  scale = ggplot2::scale_color_hue(aesthetics = "axis_color")
)

Arguments

plt  ggplot2::ggplot() object
label_data_x  data.frame() containing the mapping between x-axis labels and aes_label
label_data_y  data.frame() containing the mapping between y-axis labels and aes_label
aes_label  character or bare symbol giving the column in label_data to be mapped
scale  ggplot2 discrete color

Value

plt with axis text modified

Examples

require(ggplot2)
require(dplyr)
plt = ggplot(mpg, aes(x = manufacturer, y = drv)) + geom_jitter()
label_data = mpg %>% select(manufacturer) %>% unique() %>%
  mutate(euro = manufacturer %in% c('audi', 'volkswagen'))
map_axis_labels(plt, label_data_x = label_data, aes_label = euro)
Generate a list of tables representing clusters paired in cells

Description

A contingency table of every combination of cluster_idx up to table_order is generated. Combinations that are found in at least min_expansion number of cells are reported. All cells that have these combinations are returned, as well as cells that only have orphan_level of matching cluster_idx.

Usage

pairing_tables(
  ccdb,
  ranking_key = "grp_rank",
  table_order = 2,
  min_expansion = 2,
  orphan_level = 1,
  cluster_keys = character(),
  cluster_whitelist = NULL,
  cluster_blacklist = NULL
)

Arguments

ccdb ContigCellDB
ranking_key field in ccdb$contig_tbl giving the ranking of each contig per cell. Probably generated by a call to rank_prevalence_ccdb() or rank_chain_ccdb().
table_order Integer larger than 1. What order of cluster_idx will be paired, eg, order = 2 means that the first and second highest ranked contigs will be sought and paired in each cell
min_expansion the minimal number of times a pairing needs to occur for it to be reported
orphan_level Integer in interval [1, table_order]. Given that at least min_expansion cells are found that have table_order chains identical, how many cluster_idx pairs will we match on to select other cells. Example: orphan_level=1 means that cells that share just a single chain with an expanded pair will be reported.
cluster_keys optional character naming additional columns in ccdb$cluster_tbl to be reported in the pairing
cluster_whitelist a table of pairings or clusters that should always be reported. Here the clusters must be named "cluster_idx.1", "cluster_idx.2" (if order-2 pairs are being selected) rather than with 'ccdb$cluster_pk"
cluster_blacklist a table of pairings or clusters that will never be reported. Must be named as per cluster_whitelist.
pairing_tables

Details

For example, if `table_order=2` and `min_expansion=2` then heavy/light or alpha/beta pairs found two or more times will be returned (as well as alpha-alpha pairs, etc, if those are present). If `orphan_level=1` then all cells that share just a single chain with an expanded clone will be returned.

The `cluster_idx.1_fct` and `cluster_idx.2_fct` fields in `cell_tbl`, `idx1_tbl`, `idx2_tbl` are cast to factors and ordered such that pairings will tend to occur along the diagonal when they are cross-tabulated. This facilitates plotting.

Value

list of tables. The `cell_tbl` is keyed by the `cell_id`, with fields "cluster_idx.1", "cluster_idx.2", etc, IDing the contigs present in each cell. "cluster_idx.1_fct" and "cluster_idx.2_fct" cast these fields to factors and are reordered to maximize the number of pairs along the diagonal. The `idx1_tbl` and `idx2_tbl` report information (passed in about the `cluster_idx` by `feature_tbl`.)

The `cluster_pair_tbl` reports all pairings found of contigs, and the number of times observed.

See Also

`rank_prevalence_ccdb()`

Examples

```
library(dplyr)
tbl = tibble(clust_idx = g1(3, 2), cell_idx = rep(1:3, times = 2), contig_idx = 1:6)
ccdb = ContigCellDB(tbl, contig_pk = c('cell_idx', 'contig_idx'),
  cell_pk = 'cell_idx', cluster_pk = 'clust_idx')
  # add `grp_rank` to ccdb$contig_tbl indicating how frequent a cluster is
ccdb = rank_prevalence_ccdb(ccdb, tie_break_keys = character())
  # using `grp_rank` to determine pairing
  # no pairs found twice
pt1 = pairing_tables(ccdb)
  # all pairs found, found once.
pt2 = pairing_tables(ccdb, min_expansion = 1)
  # all pairs found, found twice.
pt2%$cell_tbl
  tbl2 = bind_rows(tbl, tbl %>% mutate(cell_idx = rep(4:6, times = 2)))
ccdb2 = ContigCellDB(tbl2, contig_pk = c('cell_idx', 'contig_idx'), cell_pk = 'cell_idx',
  cluster_pk = 'clust_idx')
  #> rank_prevalence_ccdb(tie_break_keys = character())
  # all pairs found twice
pt3 = pairing_tables(ccdb2, min_expansion = 1)
  # all pairs found, found twice.
pt3%$cell_tbl
ccdb2%$contig_tbl = ccdb2%$contig_tbl %>%
  mutate(umis = 1, reads = 1, chain = rep(c('TRA', 'TRB'), times = 6))
ccdb2 = rank_chain_ccdb(ccdb2, tie_break_keys = character())
pt4 = pairing_tables(ccdb2, min_expansion = 1, table_order = 2)
```
plot_cluster_factors  Visualization of pairs of cluster factor

Description
With factors, a pair of variables present in the contig_tbl and the cluster_tbl, generate and plot cross-tabs of the number of contigs, or its pearson residual.

Usage

plot_cluster_factors(
  ccdb,  # A ContigCellDB object.
factors,  # character length 2 of fields present
type = c("heatmap", "network"),  # Type of visualization, a heatmap or a node-edge network plot
statistic = c("pearson", "contigs"),  # Cluster characteristics visualized by pearson residuals or raw contig counts
ncluster = 0,  # integer. Omit factors that occur less than nclusters. For clarity of visualization.
chaintype  # Character in ccdb$contig_tbl$chain. If passed will subset contigs belonging to specified chain (IGH,IGK,IGL,TRA,TRB)
)

Arguments

ccdb  A ContigCellDB object.
factors  character length 2 of fields present
type  Type of visualization, a heatmap or a node-edge network plot
statistic  Cluster characteristics visualized by pearson residuals or raw contig counts
ncluster  integer. Omit factors that occur less than nclusters. For clarity of visualization.
chaintype  Character in ccdb$contig_tbl$chain. If passed will subset contigs belonging to specified chain (IGH,IGK,IGL,TRA,TRB)

Value
A ggraph object if type == 'network', and a ggplot object if type == 'heatmap'

See Also
canonicalize_cluster to "roll-up" additional contig variables into the 'cluster_tbl`

Examples

library(ggraph)
data(ccdb_ex)
ccdb_germline_ex = cluster_germline(ccdb_ex, segment_keys = c('v_gene', 'j_gene', 'chain'), cluster_pk = 'segment_idx')
ccdb_germline_ex = fine_clustering(cdb_germline_ex, sequence_key = 'cdr3_nt', type = 'DNA')
plot_cluster_factors(cdb_germline_ex,factors = c('v_gene','j_gene'), statistic = 'pearson', type = 'network',ncluster = 10, chaintype = 'TRB')
plot_permute_test

Description

Plot a histogram of permuted vs observed test statistic

Usage

plot_permute_test(perm_test)

## S3 method for class 'PermuteTestList'
tidy(x, ...)

## S3 method for class 'PermuteTest'
tidy(x, ...)

## S3 method for class 'PermuteTest'
print(x, ...)

## S3 method for class 'PermuteTestList'
print(x, max = 3, ...)

Arguments

perm_test PermuteTest or PermuteTestList output from cluster_permute_test()
x PermuteTestList
... ignored
max maximum number of components to print

Methods (by generic)

- tidy: return permutations run using a sequence of contrasts as a tibble
- tidy: return permutations as a tibble
- print: pretty-print
- print: pretty-print

See Also

cluster_permute_test
purity

Calculate number of cluster-subject singletons for the purposes of permutation testing

Description

Calculate number of cluster-subject singletons for the purposes of permutation testing

Usage

purity(cluster_idx, subject)

Arguments

cluster_idx  factor-like cluster variable
subject      factor-like subject

Value

average number of singletons

See Also

cluster_permute_test()

Examples

message("see example(cluster_permute_test)")

rank_prevalence_ccdb

Rank contigs, per cell, by experiment-wide prevalence of cluster_pk, which is added as the prevalence field

Description

Rank contigs, per cell, by experiment-wide prevalence of cluster_pk, which is added as the prevalence field
Usage

```
rank_prevalence_ccdb(
  ccdb,
  contig_filter_args = TRUE,
  tie_break_keys = c("umis", "reads")
)
```

```
rank_chain_ccdb(
  ccdb,
  contig_filter_args = TRUE,
  tie_break_keys = c("umis", "reads"),
  chain_key = "chain",
  contig_fields = tie_break_keys,
  chain_levels = c("IGL", "IGK", "TRA", "TRB", "IGH")
)
```

Arguments

- **ccdb**: ContigCellDB()
- **contig_filter_args**: an expression passed to `dplyr::filter()`. Unlike filter, multiple criteria must be & together, rather than using commas to separate. These act on `ccdb$contig_tbl`
- **tie_break_keys**: (optional) character naming fields in `contig_tbl` that are used sort the contig table in descending order. Used to break ties if `contig_filter_args` does not return a unique contig for each cluster
- **chain_key**: character naming the field in `contig_tbl` to be sorted on.
- **contig_fields**: Optional fields from `contig_tbl` that will be copied into the `cluster_tbl` from the canonical contig.
- **chain_levels**: an optional character vector providing the sort order of the chain column in tbl. If set to length zero, then the the ordering will be alphabetical

Value

ContigCellDB with modified `contig_tbl`

Functions

- `rank_chain_ccdb`: return a canonical contig by chain type, with TRB/IGH returned first. By default, ties are broken by umis and reads.

Examples

```
data(ccdb_ex)
ccdb_ex = cluster_germline(ccdb_ex)
rank_prev = rank_prevalence_ccdb(ccdb_ex)
rank_prev$contig_tbl
rank_chain = rank_chain_ccdb(ccdb_ex)
rank_chain$contig_tbl
```
rbind,ContigCellDB-method

Combine ContigCellDB along rows (contigs, cells or clusters).

Description

The union of the rows in each of the objects is taken, thus removing any rows that has an exact duplicate. This includes all fields, not just the primary key for that table. The union of the various primary keys is taken.

Usage

```r
## S4 method for signature 'ContigCellDB'
rbind(..., deparse.level = 1)
```

Arguments

- `...` ContigCellDB()
- `deparse.level` ignored

Value

ContigCellDB()

Examples

```r
data(ccdb_ex)
splat = split_cdb(ccdb_ex, 'chain', 'contig_tbl')
unite = equalize_ccdb(rbind(splat$TRA, splat$TRB), sort = TRUE)
stopifnot(all.equal(unite, ccdb_ex))
```

---

reexports

*Turn an object into a tidy tibble*

Description

Turn an object into a tidy tibble

Usage

```r
tidy(x, ...)
```

Arguments

- `x` An object to be converted into a tidy `tibble::tibble()`.
- `...` Additional arguments to tidying method.
Value

A `tibble::tibble()` with information about model components.

Methods

No methods found in currently loaded packages.

Description

Perform a `dplyr::left_join()` but check for non-key overlapping fields. If `overwrite = TRUE`, then shared columns will pull from `x` otherwise a suffix will be added to `y`. To perform this check, `by` must be specified, and it is an error if it is not.

Usage

```r
right_join_warn(...)  
left_join_warn(x, y, by, overwrite = FALSE, join = left_join, ...)  
```

Arguments

```r
... passed to joining function  
x A pair of data frames, data frame extensions (e.g. a tibble), or lazy data frames (e.g. from dbplyr or dtplyr). See Methods, below, for more details.  
y A pair of data frames, data frame extensions (e.g. a tibble), or lazy data frames (e.g. from dbplyr or dtplyr). See Methods, below, for more details.  
by character specifying columns in `x` and `y` to key on.  
overwrite logical – should non-key fields in `y` be overwritten using `x`, or should a suffix (".y") be added  
join function giving the type of join to perform, eg, left, right, inner, outer.  
```

Value
data.frame or tibble

Functions

- `right_join_warn`: perform a `dplyr::right_join()`

Examples

```r
left_join_warn(mtcars, mtcars, by = 'mpg')  
left_join_warn(mtcars, mtcars, by = 'mpg', overwrite = TRUE)  
```
split_cdb

Split into a list of `ContigCellDB()` by named fields

**Description**

Split into a list of `ContigCellDB()` by named fields

**Usage**

```r
split_cdb(ccdb, fields, tbl = "contig_tbl", drop = FALSE, equalize = TRUE)
```

**Arguments**

- `ccdb` `ContigCellDB()`
- `fields` character naming fields in `tbl`
- `tbl` one of `contig_tbl`, `cell_tbl` or `cluster_tbl`
- `drop` logical indicating if levels that do not occur should be dropped (if `f` is a factor or a list).
- `equalize` logical. Should the contig, cells and clusters be equalized by taking the intersection of their common keys?

**Value**

list of `ContigCellDB`

**Examples**

```r
data(ccdb_ex)
splat = split_cdb(ccdb_ex, 'chain', 'contig_tbl')
stopifnot(all(splat$TRA$contig_tbl$chain == 'TRA'))
stopifnot(all(splat$TRB$contig_tbl$chain == 'TRB'))
```

---

[[,ContigCellDB,character,missing-method

data.frame-like mutation/accessor generics for `ContigCellDB` objects

**Description**

A `ContigCellDB` pretend to be a `cell_tbl` data.frame in several regards. This is to enable nesting `ContigCellDB` objects in the `colData` of a `SingleCellExperiment` and so that various plotting functionality in scater can do something sensible.
Usage

## S4 method for signature 'ContigCellDB,character,missing'
x[[i, j, ...]]

## S4 method for signature 'ContigCellDB,ANY,missing,ANY'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'ContigCellDB'
dim(x)

## S4 method for signature 'ContigCellDB'
dimnames(x)

## S4 method for signature 'ContigCellDB'
nrow(x)

## S4 method for signature 'ContigCellDB'
col(x)

Arguments

- **x**: ContigCellDB
- **i**: integer or character index
- **j**: ignored
- **...**: ignored
- **drop**: ignored

Details

If `x` a ContigCellDB, then `dim(x)` and `dimnames(x)` return `dim(x$cell_tbl)` and `dimnames(x$cell_tbl)`, respectively, and `x[[col]]` returns `x$cell_tbl[[col]]`. Likewise indexing with `x[i,]` returns cells indexed by i. Finally `as.data.frame(x)` returns `x$cell_tbl`.

Value

See details.

Examples

data(ccdb_ex)
ccdb_ex[1:10,]
head(ccdb_ex[['barcode']])
dim(ccdb_ex)
dimnames(ccdb_ex)
Access public members of ContigCellDB object.

Description
Modification to members will trigger various forms of equalization. See `equalize_ccdb()` for details.

Usage
```r
## S4 method for signature 'ContigCellDB'
x$name

## S4 replacement method for signature 'ContigCellDB'
x$name <- value
```

Arguments
- **x**: A ContigCellDB object
- **name**: a slot of a ContigCellDB object (one of c('contig_tbl', 'cell_tbl', 'contig_pk', 'cell_pk', 'cluster_tbl', 'cluster_pk'))
- **value**: The value assigned to a slot of ContigCellDB object

Value
Update or return a slot of `ContigCellDB()`

See Also
- `equalize_ccdb()`

Examples
```r
data(ccdb_ex)
ccdb_ex$contig_tbl
ccdb_ex$cell_tbl
ccdb_ex$cluster_tbl
data(ccdb_ex)
ccdb_ex$contig_pk = c("sample","barcode","contig_id") # 'pop' is technically redundant with 'sample'
# Take a subset of ccdb_ex
ccdb_ex
ccdb_ex$contig_tbl = dplyr::filter(ccdb_ex$contig_tbl, pop == 'b6')
ccdb_ex
```
Index

* datasets
  - ccdb_ex, 7
  - contigs qc, 18
  - cluster_permute_test, 3
  - ContigCellDB, ANY, missing, ANY-method (ContigCellDB, character, missing-method, 38)
  - ContigCellDB, ANY, missing-method (ContigCellDB, character, missing-method, 38)
  - ContigCellDB, character, missing-method, 38
  - $, ContigCellDB-method, 40
  - $<-, ContigCellDB-method ($, ContigCellDB-method, 40

Biostrings::stringDist(), 26

canonicalize_cell, 4
canonicalize_cell(), 6
canonicalize_cluster, 5
canonicalize_cluster(), 5
ccdb.ex, 7
cdbdb.join, 7
ccdhit, 8, 10
ccdhit(), 9, 10
cdhit.ccdcb, 9
cland, 10
cluster_filterset, 11
cluster_filterset(), 15
cluster_germline, 12
cluster_logistic_test (cluster_test_by), 15
cluster_permute_test, 12
cluster_permute_test(), 34
cluster_plot, 14
ccluster_test_by, 15
ContigCellDB, 16
ContigCellDB(), 4–6, 8–10, 12, 15, 17, 21, 23, 24, 26, 35, 36, 38, 40
ContigCellDB-mutate ($, ContigCellDB-method, 40
ContigCellDB_10XVDJ (ContigCellDB), 16
contigs_qc, 7, 18
cross_tab_tbl, 19
crosstab_by_celltype, 19
crosstab_by_celltype(), 27
data.frame(), 29
dim, ContigCellDB-method (ContigCellDB, character, missing-method), 38
dimnames, ContigCellDB-method (ContigCellDB, character, missing-method), 38
dplyr::filter(), 4, 6, 16, 23, 35
dplyr::mutate(), 23
dimension, 11
enumerate_pairing (ig_chain_recode), 28
equalize_ccdb, 21
equalize_ccdb(), 40
fancy_name_contigs, 22
filter_cdb, 23
filter_cdb(), 17
fine_cluster_seqs, 24, 25
fine_clustering, 24
generate_pseudobulk, 26
ggplot2::ggplot(), 29
guess_celltype, 27
guess_celltype(), 19
hclust(), 26
hushWarning, 27
ig_chain_recode, 28
left_join_warn (right_join_warn), 37
left_join_warn(), 6
map_axis_labels, 29
modal_category (entropy), 20
mutate_cdb (filter_cdb), 23
mutate_cdb(), 17

ncol,ContigCellDB-method
(\[\[,ContigCellDB,character,missing-method), 38

np (entropy), 20
nrow,ContigCellDB-method
(\[\[,ContigCellDB,character,missing-method), 38

order(), 21

pairing_tables, 30
plot_cluster_factors, 32
plot_permute_test, 33
print.PermuteTest (plot_permute_test), 33
print.PermuteTestList
(plot_permute_test), 33
purity, 34
purity(), 13

rank_chain_ccdb (rank_prevalence_ccdb), 34
rank_chain_ccdb(), 30
rank_prevalence_ccdb, 34
rank_prevalence_ccdb(), 30, 31
rbind,ContigCellDB-method, 36
rbind.ContigCellDB
(rbind,ContigCellDB-method), 36
rbind.ContigCellDB(), 17
reexports, 36
right_join_warn, 37

split_cdb, 38
split_cdb(), 17

tcr_chain_recode (ig_chain_recode), 28
tibble::tibble(), 36, 37
tidy (reexports), 36
tidy.PermuteTest (plot_permute_test), 33
tidy.PermuteTestList
(plot_permute_test), 33