Package: dandelionR (via r-universe)

November 25, 2026

Title Single-cell Immune Repertoire Trajectory Analysis in R

Version 1.6.0

Description dandelionR is an R package for performing single-cell immune repertoire trajectory analysis, based on the original python implementation. It provides the necessary functions to interface with scRepertoire and a custom implementation of an absorbing Markov chain for pseudotime inference, inspired by the Palantir Python package.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

biocViews Software, ImmunoOncology, SingleCell

Collate 'check.R' 'constructMarkovChain.R' 'dandelionR.R' 'data.R' 'determMultiscaleSpace.R' 'terminalStateFromMarkovChain.R' 'differentiationProbabilities.R' 'filterCells.R' 'getPbs.R' 'projectProbability.R' 'maxMinSampling.R' 'minMaxScale.R' 'markovProbability.R' 'miloUmap.R' 'projectPseudotimeToCell.R' 'setupVdjPseudobulk.R' 'splitCTgene.R' 'vdjPseudobulk.R'

Imports BiocGenerics, bluster, destiny, igraph, MASS, Matrix, methods, miloR, purrr, rlang, S4Vectors, SingleCellExperiment, spam, stats, SummarizedExperiment, uwot, RANN

Suggests BiocStyle, fields, knitr, rmarkdown, RColorBrewer, scater, scRepertoire, DelayedMatrixStats, slingshot, testthat

VignetteBuilder knitr

URL https://www.github.com/tuonglab/dandelionR/

BugReports https://www.github.com/tuonglab/dandelionR/issues

Depends R (>= 4.4.0)

Config/pak/sysreqs
libfontconfig1-dev libfreetype6-dev libglpk-dev make libicu-dev libxml2-dev zlib1g-dev

Repository Bioconductor 3.24

Date/Publication 2026-11-23 22:53:40 UTC

RemoteUrl https://github.com/bioc/dandelionR

RemoteRef RELEASE_3_24

RemoteSha d4a75919f8175922f14639fa0c241596d59f75f9

Contents

demo_airr 2
demo_sce 3
differentiationProbabilities 4
markovProbability 5
miloUmap 7
project_single_value 8
projectProbability 9
projectPseudotimeToCell 9
sce_vdj 11
setupVdjPseudobulk 12
vdjPseudobulk 15

Index **18**

`demo_airr` *Example AIRR Dataset for V(D)J Analysis*

Description

The `demo_airr` object is a list of AIRR data frames from a down-sampled demo dataset derived from Suo et al., 2024, *Nature Biotechnology*.
This dataset is used in vignettes to demonstrate workflows for V(D)J analysis.
For details, see the original publication at https://www.nature.com/articles/s41587-023-01734-7.
The original files are available at https://github.com/zktuong/dandelion-demo-files.

Usage

```
data(demo_airr)
```

Format

A `SingleCellExperiment` object with the following slots:

`list` List of `DataFrames` containing the standardised AIRR data for each sample.
For information of AIRR rearrangements, see the AIRR Community standards at https://docs.airr-community.org/.

Source

Suo et al., 2024, *Nature Biotechnology*.
https://www.nature.com/articles/s41587-023-01734-7.

Examples

```
data(demo_airr)
```

---

demo_sce *Example SCE Dataset that does not contain V(D)J information*

Description

The demo_sce object is a down-sampled demo dataset derived from Suo et al., 2024, *Nature Biotechnology*.
This dataset is used in vignettes to demonstrate workflows for V(D)J analysis.
For details, see the original publication at https://www.nature.com/articles/s41587-023-01734-7.
The original Lymphoid cells data in h5ad format is available at https://developmental.cellatlas.io/fetal-immune.

Usage

```
data(demo_sce)
```

Format

A SingleCellExperiment object with the following slots:

colData A minimall DataFrame containing metadata about each sample, corresponding to obs in AnnData (Python). The following columns are relevant for vignette usage:

anno_lvl_2_final_clean Cell type annotations.

int_colData A DataFrame containing additional assay metadata important for further analysis. Includes:

- X_scvi: A dimensionality reduction matrix from the scVI model.
- UMAP: A UMAP reduction matrix.

Source

Suo et al., 2024, *Nature Biotechnology*.
https://www.nature.com/articles/s41587-023-01734-7.

Examples

```
data(demo_sce)
```

differentiationProbabilities
Compute Branch Probabilities Using Markov Chain

Description

This function calculates branch probabilities for differentiation trajectories based on a Markov chain constructed from waypoint data and pseudotime ordering.

Usage

```
differentiationProbabilities(
  wp_data,
  terminal_states = NULL,
  knn = 30L,
  pseudotime,
  waypoints,
  verbose = TRUE,
  use_RANN = TRUE
)
```

Arguments

wp_data	A multi-scale data matrix or data frame representing the waypoints.
terminal_states	Integer vector. Indices of the terminal states. Default is NULL.
knn	Integer. Number of nearest neighbors for graph construction. Default is 30L.
pseudotime	Numeric vector. Pseudotime ordering of cells.
waypoints	Integer vector. Indices of selected waypoints used to construct the Markov chain.
verbose	Boolean, whether to print messages/warnings.
use_RANN	parameter to make user choose whether to use RANN to construct Markov chain, or keep using bluster

Value

A numeric matrix or data frame containing branch probabilities for each waypoint.

`markovProbability` *Markov Chain Construction and Probability Calculation*

Description

This function preprocesses data, constructs a Markov chain, and calculates transition probabilities based on pseudotime information.

Usage

```
markovProbability(
  milo,
  diffusionmap,
  terminal_state = NULL,
  root_cell,
  knn = 30L,
  diffusiontime = NULL,
  pseudotime_key = "pseudotime",
  scale_components = TRUE,
  num_waypoints = 500,
  n_eigs = NULL,
  verbose = TRUE,
  use_RANN = TRUE
)
```

Arguments

`milo`	A `Milo` or `SingleCellExperiment` object. This object should have pseudotime stored in `colData`, which will be used to calculate probabilities. If pseudotime is available in `milo`, it takes precedence over the value provided through the `diffusiontime` parameter.
`diffusionmap`	A `DiffusionMap` object corresponding to the `milo` object. Used for Markov chain construction.
`terminal_state`	Integer. The index of the terminal state in the Markov chain.
`root_cell`	Integer. The index of the root state in the Markov chain.
`knn`	Integer. The number of nearest neighbors for graph construction. Default is `30L`.
`diffusiontime`	Numeric vector. If pseudotime is not stored in `milo`, this parameter can be used to provide pseudotime values to the function.
`pseudotime_key`	Character. The name of the column in `colData` that contains the inferred pseudotime.
`scale_components`	Logical. If `TRUE`, the components will be scaled before constructing the Markov chain. Default is `FALSE`.
`num_waypoints`	Integer. The number of waypoints to sample when constructing the Markov chain. Default is `500L`.

n_eigs	integer, default is NULL. Number of eigen vectors to use. • If is not specified, the number of eigen vectors will be determined using the eigen gap.
verbose	Logical. If TRUE, print progress. Default is TRUE.
use_RANN	parameter to make user choose whether to use RANN to construct Markov chain, or keep using bluster

Value

milo or SinglCellExperiment object with pseudotime, probabilities in its colData

Examples

```
data(sce_vdj)
# downsample to first 2000 cells
sce_vdj <- sce_vdj[, 1:2000]
sce_vdj <- setupVdjPseudobulk(sce_vdj,
    already.productive = FALSE,
    allowed_chain_status = c("Single pair", "Extra pair")
)
# Build Milo Object
set.seed(100)
milo_object <- miloR::Milo(sce_vdj)
milo_object <- miloR::buildGraph(milo_object,
    k = 50, d = 20,
    reduced.dim = "X_scvi"
)
milo_object <- miloR::makeNhoods(milo_object,
    reduced_dims = "X_scvi",
    d = 20
)

# Construct Pseudobulked VDJ Feature Space
pb.milo <- vdjPseudobulk(milo_object, col_to_take = "anno_lvl_2_final_clean")
pb.milo <- scater::runPCA(pb.milo, assay.type = "Feature_space")

# Define root and branch tips
pca <- t(as.matrix(SingleCellExperiment::reducedDim(pb.milo, type = "PCA")))
branch.tips <- c(which.min(pca[, 2]), which.max(pca[, 2]))
names(branch.tips) <- c("CD8+T", "CD4+T")
root <- which.min(pca[, 1])

# Construct Diffusion Map
dm <- destiny::DiffusionMap(t(pca), n_pcs = 10, n_eigs = 5)
dif.pse <- destiny::DPT(dm, tips = c(root, branch.tips), w_width = 0.1)

# Markov Chain Construction
pb.milo <- markovProbability(
    milo = pb.milo,
    diffusionmap = dm,
    diffusiontime = dif.pse[[paste0("DPT", root)]],
```

```
    terminal_state = branch.tips,
    root_cell = root,
    pseudotime_key = "pseudotime"
)
```

`miloUmap` *Perform UMAP on the Adjacency Matrix of a Milo Object*

Description

This function uses `uwot::umap` to perform UMAP dimensionality reduction on the adjacency matrix of the KNN graph in a Milo object.

Usage

```
miloUmap(
  milo,
  slot_name = "UMAP_knngraph",
  n_neighbors = 50L,
  metric = "euclidean",
  min_dist = 0.3,
  use_graph = TRUE,
  ...
)
```

Arguments

`milo`	the milo object with knn graph that needed to conduct umap on.
`slot_name`	character, with default 'UMAP_knngraph'. • The slot name in reduceDim where the result store
`n_neighbors`	integer, with default 50L. • the size of local neighborhood (in terms of number of neighboring sample points) used for manifold approximation. • Here, the goal is to create large enough neighborhoods to capture the local manifold structure to allow for hypersampling.
`metric`	character, with default 'euclidean' • the choice of metric used to measure distance to find nearest neighbors. Default is 'euclidean'.
`min_dist`	numeric, with default 0.3 • the minimum distance between points in the low dimensional space
`use_graph`	Logical, default TRUE. • Whether to run UMAP on the graph adjacency matrix (TRUE) as in Dandelion, or directly on the latent space (FALSE) for faster performance.
`...`	other parameters passed to uwot::umap

Value

milo object with umap reduction

Examples

```
data(sce_vdj)
# downsample to just 1000 cells
sce_vdj <- sce_vdj[, 1:1000]
sce_vdj <- setupVdjPseudobulk(sce_vdj,
    already.productive = FALSE,
    allowed_chain_status = c("Single pair", "Extra pair")
)
# Build Milo Object
milo_object <- miloR::Milo(sce_vdj)
milo_object <- miloR::buildGraph(milo_object,
    k = 50, d = 20,
    reduced.dim = "X_scvi"
)
milo_object <- miloR::makeNhoods(milo_object,
    reduced_dims = "X_scvi", d = 20
)

# Construct UMAP on Milo Neighbor Graph
milo_object <- miloUmap(milo_object)
```

---

project_single_value *Function to project pseudobulk-level values to single-cell level*

Description

Function to project pseudobulk-level values to single-cell level

Usage

```
project_single_value(x, y, value_name, verbose = TRUE)
```

Arguments

x	Numeric vector, pseudobulk-level value to be projected.
y	Matrix (pseudobulk x cell), used to project x back to cell level.
value_name	Character, name of the value being projected.
verbose	Boolean, whether to print messages/warnings.

Value

Numeric vector of projected values at cell level.

`projectProbability` *Project Probabilities from Markov Chain to Pseudobulks*

Description

This function projects probabilities calculated from a Markov chain onto each pseudobulk based on a diffusion distance matrix.

Usage

```
projectProbability(
  diffusionmap,
  waypoints,
  probabilities,
  t = 1,
  verbose = TRUE
)
```

Arguments

`diffusionmap`	diffusion map, used to reconstruct diffustion distance matrix
`waypoints`	Integer vector. Indices of the waypoints used in the Markov chain.
`probabilities`	Numeric vector. Probabilities associated with the waypoints, calculated from the Markov chain.
`t`	Numeric. The diffusion time to be used in the projection.
`verbose`	Boolean, whether to print messages/warnings.

Value

each pseudobulk's probabilites

`projectPseudotimeToCell` *Project Pseudotime and Branch Probabilities to Single Cells*

Description

This function projects pseudotime and branch probabilities from pseudobulk data to single-cell resolution (`milo`). The results are stored in the `colData` of the `milo` object.

Usage

```
projectPseudotimeToCell(
  milo,
  pb_milo,
  value_key = NULL,
  suffix = "",
  verbose = TRUE
)
```

Arguments

milo	A SingleCellExperiment or Milo object. Represents single-cell data where pseudotime and branch probabilities will be projected.
pb_milo	A pseudobulk Milo object. Contains aggregated branch probabilities and pseudotime information to be transferred to single cells.
value_key	Character. The column name in colData of pb_milo that contains the value that is needed to be projected back. Default is NULL.
suffix	Character. A suffix to be added to the new column names in colData. Default is an empty string ('').
verbose	Boolean, whether to print messages/warnings.

Value

subset of milo or SingleCellExperiment object where cell that do not belong to any neighbourhood are removed and projected pseudotime information stored colData

Examples

```
data(sce_vdj)
# downsample to first 2000 cells
sce_vdj <- sce_vdj[, 1:2000]
sce_vdj <- setupVdjPseudobulk(sce_vdj,
    already.productive = FALSE,
    allowed_chain_status = c("Single pair", "Extra pair")
)
# Build Milo Object
set.seed(100)
milo_object <- miloR::Milo(sce_vdj)
milo_object <- miloR::buildGraph(milo_object,
    k = 50, d = 20,
    reduced.dim = "X_scvi"
)
milo_object <- miloR::makeNhoods(milo_object,
    reduced_dims = "X_scvi",
    d = 20
)

# Construct Pseudobulked VDJ Feature Space
pb.milo <- vdjPseudobulk(milo_object, col_to_take = "anno_lvl_2_final_clean")
```

```
pb.milo <- scater::runPCA(pb.milo, assay.type = "Feature_space")

# Define root and branch tips
pca <- t(as.matrix(SingleCellExperiment::reducedDim(pb.milo, type = "PCA")))
branch.tips <- c(which.min(pca[, 2]), which.max(pca[, 2]))
names(branch.tips) <- c("CD8+T", "CD4+T")
root <- which.min(pca[, 1])

# Construct Diffusion Map
dm <- destiny::DiffusionMap(t(pca), n_pcs = 10, n_eigs = 5)
dif.pse <- destiny::DPT(dm, tips = c(root, branch.tips), w_width = 0.1)

# Markov Chain Construction
pb.milo <- markovProbability(
    milo = pb.milo,
    diffusionmap = dm,
    diffusiontime = dif.pse[[paste0("DPT", root)]],
    terminal_state = branch.tips,
    root_cell = root,
    pseudotime_key = "pseudotime"
)
# Project Pseudobulk Data
projected_milo <- projectPseudotimeToCell(
    milo_object,
    pb.milo,
    value_key = c("pseudotime", "CD8+T", "CD4+T")
)
```

`sce_vdj` *Example Dataset for V(D)J Analysis*

Description

The `sce_vdj` object is a down-sampled demo dataset derived from Suo et al., 2024, *Nature Biotechnology*.
This dataset is used in vignettes to demonstrate workflows for V(D)J analysis.
For details, see the original publication at https://www.nature.com/articles/s41587-023-01734-7.

Usage

```
data(sce_vdj)
```

Format

A `SingleCellExperiment` object with the following slots:

`colData` A `DataFrame` containing metadata about each sample, corresponding to `obs` in AnnData (Python). The following columns are relevant for vignette usage:

`productive_(mode)_VDJ`, `productive_(mode)_VJ` Factors indicating whether the heavy or light chain is productive. `mode` refers to the extraction mode for V(D)J genes and can be one of:

- `'abT'`: TCR alpha-beta
- `'gdT'`: TCR gamma-delta
- `'B'`: BCR

`Gene segment fields` Gene segment annotations with column names in the format `(v/d/j)_call_(mode)_(VDJ/VJ)`. Examples include:

- `v_call_abT_VDJ`: V gene for TCR alpha-beta VDJ recombination
- `d_call_abT_VJ`: D gene for TCR alpha-beta VJ recombination

`chain_status` A factor describing the receptor chain's status.

`anno_lvl_2_final_clean` Cell type annotations.

`int_colData` A `DataFrame` containing additional assay metadata important for further analysis. Includes:

- `X_scvi`: A dimensionality reduction matrix from the scVI model.
- `UMAP`: A UMAP reduction matrix.

Source

Suo et al., 2024, *Nature Biotechnology*.
https://www.nature.com/articles/s41587-023-01734-7.

Examples

```
data(sce_vdj)
```

`setupVdjPseudobulk` *Preprocess V(D)J Data for Pseudobulk Analysis*

Description

This function preprocesses single-cell V(D)J sequencing data for pseudobulk analysis. It filters data based on productivity and chain status, subsets data, extracts main V(D)J genes, and removes unmapped entries.

Usage

```
setupVdjPseudobulk(
  sce,
  mode_option = c("abT", "gdT", "B"),
  already.productive = TRUE,
  productive_cols = NULL,
  productive_vj = TRUE,
  productive_vdj = TRUE,
  allowed_chain_status = NULL,
```

```
  subsetby = NULL,
  groups = NULL,
  extract_cols = NULL,
  filter_unmapped = TRUE,
  check_vj_mapping = c(TRUE, TRUE),
  check_vdj_mapping = c(TRUE, FALSE, TRUE),
  check_extract_cols_mapping = NULL,
  remove_missing = TRUE,
  verbose = TRUE
)
```

Arguments

sce	A SingleCellExperiment object. V(D)J data should be contained in colData for filtering.
mode_option	Optional character. Specifies the mode for extracting V(D)J genes. If NULL, extract_cols must be specified. Default is NULL.
already.productive	Logical. Whether the data has already been filtered for productivity. If TRUE, skips productivity filtering. Default is FALSE.
productive_cols	Character vector. Names of colData columns used for productivity filtering. Default is NULL.
productive_vj	Logical. If TRUE, retains cells where the main VJ chain is productive. Default is TRUE.
productive_vdj	Logical. If TRUE, retains cells where the main VDJ chain is productive. Default is TRUE.
allowed_chain_status	Character vector. Specifies chain statuses to retain. Valid options include`c('single pair', 'Extra pair', 'Extra pair-exception','Orphan VDJ', 'Orphan VDJ-exception')`. Default is NULL.
subsetby	Character. Name of a colData column for subsetting. Default is NULL.
groups	Character vector. Specifies the subset condition for filtering. Default is NULL.
extract_cols	Character vector. Names of colData columns where V(D)J information is stored, used instead of the standard columns. Default is NULL.
filter_unmapped	Logic. Whether to filter unmapped data. Default is TRUE.
check_vj_mapping	Logic vector. Whether to check for VJ mapping. Default is c(TRUE, TRUE). • If the first element is TRUE, function will filter the unmapped data in V gene of the VJ chain • If the second element is TRUE, function will filter the unmapped data in J gene of the VJ chain
check_vdj_mapping	Logic vector. Specifies columns to check for VDJ mapping. Default is c(TRUE, FALSE, 'TRUE).

- If the first element is TRUE, function will filter the unmapped data in V gene of the VDJ chain
- If the second element is TRUE, function will filter the unmapped data in D gene of the VDJ chain
- If the third element is TRUE, function will filter the unmapped data in J gene of the VDJ chain

`check_extract_cols_mapping`
: Character vector. Specifies columns related to `extract_cols` for mapping checks. Default is `NULL`.

`remove_missing`
: Logical. If `TRUE`, removes cells with contigs matching the filter. If `FALSE`, masks them with uniform values. Default is `TRUE`.

`verbose`
: Logical. Whether to print messages. Default is `TRUE`.

Details

The function performs the following preprocessing steps:

- **Productivity Filtering**:
  - Skipped if `already.productive = TRUE`.
  - Filters cells based on productivity using `productive_cols` or standard `colData` columns named `productive_{mode_option}_{type}` (where `type` is 'VDJ' or 'VJ').
  - *mode_option*
    - function will check colData(s) named `productive_{mode_option}_{type}`, where type should be 'VDJ' or 'VJ' or both, depending on values of productive_vj and productive_vdj.
    - If set as `NULl`, the function needs the option 'extract_cols' to be specified
  - *productive_cols*
    - must be be specified when productivity filtering is need to conduct and mode_option is NULL.
    - where VDJ/VJ information is stored so that this will be used instead of the standard columns.
  - *productive_vj, productive_vdj*
    - If `TRUE`, cell will only be kept if the main V(D)J chain is productive
- **Chain Status Filtering**:
  - Retains cells with chain statuses specified by `allowed_chain_status`.
- **Subsetting**:
  - Conducted only if both `subsetby` and `groups` are provided.
  - Retains cells matching the `groups` condition in the `subsetby` column.
- **Main V(D)J Extraction**:
  - Uses `extract_cols` to specify custom columns for extracting V(D)J information.
- **Unmapped Data Filtering**:
  - decided to removes or masks cells based on `filter_unmapped`.
  - Checks specific columns for unclear mappings using `check_vj_mapping`, `check_vdj_mapping`, or `check_extract_cols_mapping`.

- *filter_unmapped*
  - pattern to be filtered from object.
  - If is set to be NULL, the filtering process will not start
- *check_vj_mapping, check_vdj_mapping*
  - only colData specified by these arguments (check_vj_mapping and check_vdj_mapping) will be checked for unclear mappings
- *check_extract_cols_mapping, related to extract_cols*
  - Only colData specified by the argument will be checked for unclear mapping, the colData should first specified by extract_cols
- remove_missing
  - If TRUE, will remove cells with contigs matching the filter from the object.
  - If FALSE, will mask them with a uniform value dependent on the column name.

Value

filtered SingleCellExperiment object

Examples

```
# load data
data(sce_vdj)
# check the dimension
dim(sce_vdj)
# filtered the data
sce_vdj <- setupVdjPseudobulk(
    sce = sce_vdj,
    mode_option = "abT", # set the mode to alpha-beta TCR
    allowed_chain_status = c("Single pair", "Extra pair"),
    already.productive = FALSE
) # need to filter the unproductive cells
# check the remaining dim
dim(sce_vdj)
```

---

vdjPseudobulk *Generate Pseudobulk V(D)J Feature Space*

Description

This function creates a pseudobulk V(D)J feature space from single-cell data, aggregating V(D)J information into pseudobulk groups. It supports input as either a Milo object or a SingleCellExperiment object.

Usage

```
vdjPseudobulk(
  milo,
  pbs = NULL,
  col_to_bulk = NULL,
 extract_cols = c("v_call_abT_VDJ_main", "j_call_abT_VDJ_main", "v_call_abT_VJ_main",
    "j_call_abT_VJ_main"),
  mode_option = c("abT", "gdT", "B"),
  col_to_take = NULL,
  normalise = TRUE,
  renormalize = FALSE,
  min_count = 1L,
  verbose = TRUE
)
```

Arguments

milo	A Milo or SingleCellExperiment object containing V(D)J data.
pbs	Optional. A binary matrix with cells as rows and pseudobulk groups as columns. • If milo is a Milo object, this parameter is not required. • If milo is a SingleCellExperiment object, either pbs or col_to_bulk must be provided.
col_to_bulk	Optional character or character vector. Specifies colData column(s) to generate pbs. If multiple columns are provided, they will be combined. Default is NULL. • If milo is a Milo object, this parameter is not required. • If milo is a SingleCellExperiment object, either pbs or col_to_bulk must be provided.
extract_cols	Character vector. Specifies column names where V(D)J information is stored. Default is c('v_call_abT_VDJ_main', 'j_call_abT_VDJ_main', ' 'v_call_abT_VJ_main', 'j_ca
mode_option	Character. Specifies the mode for extracting V(D)J genes. Must be one of c('B', 'abT', 'gdT'). Default is 'abT'. • Note: This parameter is considered only when extract_cols = NULL. • If NULL, uses column names such as v_call_VDJ instead of v_call_abT_VDJ.
col_to_take	Optional character or vector of characters. Specifies names of colData of milo that need to identify the most common value for each pseudobulk Default is NULL.
normalise	Logical. If TRUE, scales the counts of each V(D)J gene group to 1 for each pseudobulk. Default is TRUE.
renormalize	Logical. If TRUE, rescales the counts of each V(D)J gene group to 1 for each pseudobulk after removing 'missing' calls. Useful when setupVdjPseudobulk() was run with remove_missing = FALSE. Default is FALSE.
min_count	Integer. Sets pseudobulk counts in V(D)J gene groups with fewer than this many non-missing calls to 0. Default is 1.
verbose	Logical. If TRUE, prints messages and warnings. Default is TRUE.

Details

This function aggregates V(D)J data into pseudobulk groups based on the following logic:

- **Input Requirements**:
- If `milo` is a `Milo` object, neither `pbs` nor `col_to_bulk` is required.
- If `milo` is a `SingleCellExperiment` object, the user must provide either `pbs` or `col_to_bulk`.
- **Normalization**:
- When `normalise = TRUE`, scales V(D)J counts to 1 for each pseudobulk group.
- When `renormalize = TRUE`, rescales the counts after removing 'missing' calls.
- **Mode Selection**:
- If `extract_cols = NULL`, the function relies on `mode_option` to determine which V(D)J columns to extract.
- **Filtering**:
- Uses `min_count` to filter pseudobulks with insufficient counts for V(D)J groups.

Value

SingleCellExperiment object

Examples

```
data(sce_vdj)
sce_vdj <- setupVdjPseudobulk(sce_vdj,
    already.productive = FALSE,
    allowed_chain_status = c("Single pair", "Extra pair")
)
# Build Milo Object
milo_object <- miloR::Milo(sce_vdj)
milo_object <- miloR::buildGraph(milo_object,
    k = 50, d = 20,
    reduced.dim = "X_scvi"
)
milo_object <- miloR::makeNhoods(milo_object,
    reduced_dims = "X_scvi",
    d = 20
)

# Construct pseudobulked VDJ feature space
pb.milo <- vdjPseudobulk(milo_object, col_to_take = "anno_lvl_2_final_clean")
```

Index

∗ **datasets**
demo_airr, 2
demo_sce, 3
sce_vdj, 11

demo_airr, 2
demo_sce, 3
differentiationProbabilities, 4

markovProbability, 5
miloUmap, 7

project_single_value, 8
projectProbability, 9
projectPseudotimeToCell, 9

sce_vdj, 11
setupVdjPseudobulk, 12

vdjPseudobulk, 15