Package 'immundata'

October 9, 2025

Title A Unified Data Layer for Large-Scale Single-Cell, Spatial and Bulk Immunomics

Version 0.0.5

Contact support@immunomind.com

Description Provides a unified data layer for single-cell, spatial and bulk T-cell and B-cell immune receptor repertoire data. Think AnnData or SeuratObject, but for AIRR data, a.k.a. Adaptive Immune Receptor Repertoire, VDJ-seq, RepSeq, or VDJ sequencing data.

License Apache License (>= 2)

URL https://immunomind.github.io/docs/, https://github.com/immunomind/immundata

BugReports https://github.com/immunomind/immundata/issues

Encoding UTF-8

RoxygenNote 7.3.3

Depends R (>= 4.1.0), dplyr, duckplyr (>= 1.1.0)

Imports checkmate, cli, dbplyr, ggplot2, glue, jsonlite (>= 2.0.0), lifecycle, R6, readr, rlang, tibble, tools, utils

Suggests rmarkdown, testthat (>= 3.0.0), Seurat

Config/testthat/edition 3

Config/testthat/parallel true

NeedsCompilation no

Author Vadim I. Nazarov [aut, cre] (ORCID: <https://orcid.org/0000-0003-3659-2709>)

Maintainer Vadim I. Nazarov <support@immunomind.com>

Repository CRAN

Date/Publication 2025-10-09 12:30:02 UTC

Contents

agg_receptors . . . 2
agg_repertoires . . . 4
annotate_immundata . . . 6
annotate_seurat . . . 9
count.ImmunData . . . 10
filter_immundata . . . 10
from_immunarch . . . 13
imd_schema . . . 15
ImmunData . . . 15
make_default_preprocessing . . . 17
make_receptor_schema . . . 18
make_seq_options . . . 19
mutate_immundata . . . 20
read_immundata . . . 22
read_metadata . . . 24
read_repertoires . . . 25
write_immundata . . . 29

Index **31**

`agg_receptors` *Aggregates AIRR data into receptors*

Description

Processes a table of immune receptor sequences (chains or clonotypes) to identify unique receptors based on a specified schema. It assigns a unique identifier (`imd_receptor_id`) to each distinct receptor signature and returns an annotated table linking the original sequence data to these receptor IDs.

This function is a core component used within `read_repertoires()` and handles different input data structures:

- Simple tables (no counts, no cell IDs).
- Bulk sequencing data (using a count column).
- Single-cell data (using a barcode/cell ID column). For single-cell data, it can perform chain pairing if the schema specifies multiple chains (e.g., TRA and TRB).

Usage

```
agg_receptors(
  dataset,
  schema,
  barcode_col = NULL,
  count_col = NULL,
  locus_col = NULL,
  umi_col = NULL
)
```

Arguments

dataset	A data frame or duckplyr_df containing sequence/clonotype data. Must include columns specified in schema and potentially barcode_col, count_col, locus_col, umi_col. Could be idata$annotations.
schema	Defines how a unique receptor is identified. Can be: • A character vector of column names representing receptor features (e.g., c("v_call", "j_call", "junction_aa")). • A list created by make_receptor_schema(), specifying both features (character vector) and optionally chains (character vector of locus names like "TRA", "TRB", "IGH", "IGK", "IGL", max length 2). Specifying chains triggers filtering by locus and enables pairing logic if two chains are given.
barcode_col	Character(1). The name of the column containing cell identifiers (barcodes). Required for single-cell processing and chain pairing. Default: NULL.
count_col	Character(1). The name of the column containing counts (e.g., UMI counts for bulk, clonotype frequency). Used for bulk data processing. Default: NULL. Cannot be specified if barcode_col is set.
locus_col	Character(1). The name of the column specifying the chain locus (e.g., "TRA", "TRB"). Required if schema includes chains for filtering or pairing. Default: NULL.
umi_col	Character(1). The name of the column containing UMI counts. Required for *paired-chain single-cell* data (length(schema$chains) == 2). Used to select the most abundant chain per locus within a cell when multiple chains of the same locus are present. Default: NULL.

Details

The function performs the following main steps:

1. **Validation:** Checks inputs, schema validity, and existence of required columns.
2. **Schema Parsing:** Determines receptor features and target chains from schema.
3. **Locus Filtering:** If schema$chains is provided, filters the dataset to include only rows matching the specified locus/loci.
4. **Processing Logic (based on** barcode_col **and** count_col**):**
   - **Simple Table/Bulk (No Barcodes):** Assigns unique internal barcode/chain IDs. Identifies unique receptors based on schema$features. Calculates imd_chain_count (1 for simple table, from count_col for bulk).
   - **Single-Cell (Barcodes Provided):** Uses barcode_col for imd_barcode_id.
     - **Single Chain:** (length(schema$chains) <= 1). Identifies unique receptors based on schema$features. imd_chain_count is 1.
     - **Paired Chain:** (length(schema$chains) == 2). Requires locus_col and umi_col. Filters chains within each cell/locus group based on max umi_col. Creates paired receptors by joining the two specified loci for each cell based on schema$features from both. Assigns a unique imd_receptor_id to each *pair*. imd_chain_count is 1 (representing the chain record).

5. **Output:** Returns an annotated data frame containing original columns plus internal identifiers (`imd_receptor_id`, `imd_barcode_id`, `imd_chain_id`) and counts (`imd_chain_count`).

Internal column names are typically managed by `immundata:::imd_schema()`.

Value

A `duckplyr_df` (or data frame) representing the annotated sequences. This table links each original sequence record (chain) to a defined receptor and includes standardized columns:

- `imd_receptor_id`: Integer ID unique to each distinct receptor signature.
- `imd_barcode_id`: Integer ID unique to each cell/barcode (or row if no barcode).
- `imd_chain_id`: Integer ID unique to each input row (chain).
- `imd_chain_count`: Integer count associated with the chain (1 for SC/simple, from `count_col` for bulk). This output is typically assigned to the `$annotations` field of an `ImmunData` object.

See Also

`read_repertoires()`, `make_receptor_schema()`, ImmunData

`agg_repertoires` *Aggregate AIRR data into repertoires*

Description

Groups the annotation table of an `ImmunData` object by user-specified columns to define distinct *repertoires* (e.g., based on sample, donor, time point). It then calculates summary statistics both per-repertoire and per-receptor within each repertoire.

Calculated **per repertoire**:

- `n_barcodes`: Total number of unique cells/barcodes within the repertoire (sum of `imd_chain_count`, effectively summing unique cells if input was SC, or total counts if input was bulk).
- `n_receptors`: Number of unique receptors (`imd_receptor_id`) found within the repertoire.

Calculated **per annotation row** (receptor within repertoire context):

- `imd_count`: Total count of a specific receptor (`imd_receptor_id`) within the specific repertoire it belongs to in that row (sum of relevant `imd_chain_count`).
- `imd_proportion`: The proportion of the repertoire's total `n_barcodes` accounted for by that specific receptor (`imd_count / n_barcodes`).
- `n_repertoires`: The total number of distinct repertoires (across the entire dataset) in which this specific receptor (`imd_receptor_id`) appears.

These statistics are added to the annotation table, and a summary table is stored in the `$repertoires` slot of the returned object.

Usage

```
agg_repertoires(idata, schema = "repertoire_id")
```

Arguments

idata	An ImmunData object, typically the output of read_repertoires() or read_immundata(). Must contain the $annotations table with columns specified in schema and internal columns like imd_receptor_id and imd_chain_count.
schema	Character vector. Column name(s) in idata$annotations that define a unique repertoire. For example, c("SampleID") or c("DonorID", "TimePoint"). Columns must exist in idata$annotations. Default: "repertoire_id" (assumes such a column exists).

Details

The function operates on the idata$annotations table:

1. **Validation:** Checks idata and existence of schema columns. Removes any pre-existing repertoire summary columns to prevent duplication.
2. **Repertoire Definition:** Groups annotations by the schema columns. Calculates total counts (n_barcodes) per group. Assigns a unique integer imd_repertoire_id to each distinct repertoire group. This forms the initial repertoires_table.
3. **Receptor Counts & Proportion:** Calculates the sum of imd_chain_count for each receptor within each repertoire (imd_count). Calculates the proportion (imd_proportion) of each receptor within its repertoire.
4. **Repertoire & Receptor Stats:** Counts unique receptors per repertoire (n_receptors, added to repertoires_table). Counts the number of distinct repertoires each unique receptor appears in (n_repertoires).
5. **Join Results:** Joins the calculated imd_count, imd_proportion, and n_repertoires back to the annotation table based on repertoire columns and imd_receptor_id.
6. **Return New Object:** Creates and returns a *new* ImmunData object containing the updated $annotations table (with the added statistics) and the $repertoires slot populated with the repertoires_table (containing schema columns, imd_repertoire_id, n_barcodes, n_receptors).

The original idata object remains unmodified. Internal column names are typically managed by immundata:::imd_schema().

Value

A **new** ImmunData object. Its $annotations table includes the added columns (imd_repertoire_id, imd_count, imd_proportion, n_repertoires). Its $repertoires slot contains the summary table linking schema columns to imd_repertoire_id, n_barcodes, and n_receptors.

See Also

read_repertoires() (which can call this function), ImmunData class.

Examples

```
## Not run:
# Assume 'idata_raw' is an ImmunData object loaded via read_repertoires
# but *without* providing 'repertoire_schema' initially.
# It has $annotations but $repertoires is likely NULL or empty.
# Assume idata_raw$annotations has columns "SampleID" and "TimePoint".

# Define repertoires based on SampleID and TimePoint
idata_aggregated <- agg_repertoires(idata_raw, schema = c("SampleID", "TimePoint"))

# Explore the results
print(idata_aggregated)
print(idata_aggregated$repertoires)
print(head(idata_aggregated$annotations)) # Note the new columns

## End(Not run)
```

`annotate_immundata` *Annotate ImmunData object*

Description

Joins additional annotation data to the annotations slot of an `ImmunData` object.

This function allows you to add extra information to your repertoire data by joining a dataframe of annotations based on specified columns. It supports joining by one or more columns.

Usage

```
annotate_immundata(
  idata,
  annotations,
  by,
  keep_repertoires = TRUE,
  remove_limit = FALSE
)

annotate(idata, annotations, by, keep_repertoires = TRUE, remove_limit = FALSE)

annotate_receptors(
  idata,
  annotations,
  annot_col = imd_schema("receptor"),
  keep_repertoires = TRUE,
  remove_limit = FALSE
)

annotate_barcodes(
```

```
  idata,
  annotations,
  annot_col = "<rownames>",
  keep_repertoires = TRUE,
  remove_limit = FALSE
)

annotate_chains(
  idata,
  annotations,
  annot_col = imd_schema("chain"),
  keep_repertoires = TRUE,
  remove_limit = FALSE
)
```

Arguments

idata	An ImmunData R6 object containing repertoire and annotation data.
annotations	A data frame containing the annotations to be joined.
by	A named character vector specifying the columns to join by. The names of the vector should be the column names in idata$annotations and the values should be the corresponding column names in the annotations data frame.
keep_repertoires	Logical. If TRUE (default) and the ImmunData object contains repertoire data (idata$schema_repertoire is not NULL), the repertoires will be re-aggregated after joining the annotations. Set to FALSE if you do not want to re-aggregate repertoires immediately.
remove_limit	Logical. If FALSE (default), a warning will be issued if the annotations data frame has 100 or more columns, suggesting potential performance issues. Set to TRUE to disable this warning and allow joining of annotations with an arbitrary number of columns. Use with caution, as joining wide dataframes can be memory-intensive and slow.
annot_col	A character vector specifying the column with receptor, barcode or chain identifiers to annotate a corresponding receptors, barode or chains in idata.

Details

The function performs a left join operation, keeping all rows from idata$annotations and adding matching columns from the annotations data frame. If there are multiple matches in annotations for a row in idata$annotations, all combinations will be returned, potentially increasing the number of rows in the resulting annotations table.

The function uses checkmate to validate the input types and structure.

A check is performed to ensure that the columns specified in by exist in both idata$annotations and the annotations data frame.

The annotations data frame is converted to a duckdb tibble internally for efficient joining, especially with large datasets.

Value

A new ImmunData object with the annotations joined to the annotations slot.

Warning

By default (remove_limit = FALSE), joining an annotations data frame with 100 or more columns will trigger a warning. This is a safeguard to prevent accidental joining of very wide data (e.g., gene expression data) that could lead to performance degradation or crashes. If you understand the risks and intend to join a wide data frame, set remove_limit = TRUE.

Examples

```
## Not run:
# Assuming 'my_immun_data' is an ImmunData object and 'sample_info' is a data frame
# with a column 'sample_id' matching 'sample' in my_immun_data$annotations
# and additional columns like 'treatment' and 'disease_status'.

sample_info <- data.frame(
  sample_id = c("sample1", "sample2", "sample3", "sample4"),
  treatment = c("Treatment A", "Treatment B", "Treatment A", "Treatment C"),
  disease_status = c("Healthy", "Disease", "Healthy", "Disease"),
  stringsAsFactors = FALSE # Important to keep characters as characters
)

# Join sample information using the 'sample' column
my_immun_data_annotated <- annotate(
  idata = my_immun_data,
  annotations = sample_info,
  by = c("sample" = "sample_id")
)

# New sample_info

# Join data by multiple columns, e.g., 'sample' and 'barcode'
# Assuming 'cell_annotations' is a data frame with 'sample_barcode' and 'cell_type'
my_immun_data_cell_annotated <- annotate(
  idata = my_immun_data,
  annotations = cell_annotations,
  by = c("sample" = "sample", "barcode" = "sample_barcode")
)

# Join a wide dataframe, suppressing the column limit warning
# Assuming 'gene_expression' is a data frame with 'barcode' and many gene columns
my_immun_data_gene_expression <- annotate(
  idata = my_immun_data,
  annotations = gene_expression,
  by = c("barcode" = "barcode"),
  remove_limit = TRUE
)

## End(Not run)
```

annotate_seurat *Annotate a Seurat object from ImmunData (by barcode)*

Description

Copy selected columns from `idata$annotations` to Seurat metadata using the cell barcode. This is the simplest way to transfer data from `immundata` to Seurat object, e.g., for plotting data on UMAP.

Usage

```
annotate_seurat(idata, sdata, cols)
```

Arguments

`idata`	An ImmunData object.
`sdata`	A Seurat object (cells are columns; barcodes are `colnames(sdata)`).
`cols`	Character vector with column names to transfer from `idata$annotations`. Typical choices: `"clonal_prop_bin"` or `"clonal_rank_bin"`.

Details

See functions `annotate_clonality_rank` and `annotate_clonality_prop` in `immunarch` package.

Value

The updated Seurat object with new metadata columns.

See Also

ImmunData, SeuratObject::AddMetaData

Examples

```
## Not run:
# After annotating receptors:
idata <- annotate_clonality_prop(idata)

# Transfer the clonality bin to Seurat and plot:
sdata <- annotate_seurat(idata, sdata, cols = "clonal_prop_bin")
Seurat::DimPlot(sdata, reduction = "umap", group.by = "clonal_prop_bin", shuffle = TRUE)

# Alternative: rank bins
idata <- annotate_clonality_rank(idata, bins = c(10, 100))
sdata <- annotate_seurat(idata, sdata, cols = "clonal_rank_bin")

## End(Not run)
```

count.ImmunData *Count the number of chains in ImmunData*

Description

Count the number of chains in ImmunData

Usage

```
## S3 method for class 'ImmunData'
count(x, ..., wt = NULL, sort = FALSE, name = NULL)
```

Arguments

x	ImmunData object.
...	Not used.
wt	Not used.
sort	Not used.
name	Not used.

filter_immundata *Filter ImmunData by receptor features, barcodes or any annotations*

Description

Provides flexible filtering options for an ImmunData object.

filter() is the main function, allowing filtering based on receptor features (e.g., CDR3 sequence) using various matching methods (exact, regex, fuzzy) and/or standard dplyr-style filtering on annotation columns.

filter_barcodes() is a convenience function to filter by specific cell barcodes.

filter_receptors() is a convenience function to filter by specific receptor identifiers.

Usage

```
filter_immundata(idata, ..., seq_options = NULL, keep_repertoires = TRUE)

## S3 method for class 'ImmunData'
filter(
  .data,
  ...,
  .by = NULL,
  .preserve = FALSE,
  seq_options = NULL,
```

```
  keep_repertoires = TRUE
)

filter_barcodes(idata, barcodes, keep_repertoires = TRUE)

filter_receptors(idata, receptors, keep_repertoires = TRUE)
```

Arguments

idata, .data	An ImmunData object.
...	For filter, these are regular dplyr-style filtering expressions (e.g., V_gene == "IGHV1-1", chain == "IGH") applied to the $annotations table *before* sequence filtering. Ignored by filter_barcodes and filter_receptors.
seq_options	For filter, an optional named list specifying sequence-based filtering options. Use make_seq_options() for convenient creation. The list can contain: • query_col (Character scalar): The name of the column in $annotations containing sequences to compare (e.g., "CDR3_aa", "FR1_nt"). • patterns (Character vector): A vector of sequences or regular expressions to match against query_col. • method (Character scalar): The matching method. One of "exact", "regex", "lev" (Levenshtein distance), or "hamm" (Hamming distance). Defaults typically handled by make_seq_options. • max_dist (Numeric scalar): For fuzzy methods ("lev", "hamm"), the maximum allowed distance. Rows with distance <= max_dist are kept. Defaults typically handled by make_seq_options. • name_type (Character scalar): Determines column names in intermediate distance calculations if applicable ("index" or "pattern"). Passed through to internal annotation functions. Defaults typically handled by make_seq_options. If seq_options is NULL (the default), no sequence-based filtering is performed.
keep_repertoires	Logical scalar. If TRUE (the default) and the input idata has repertoire information (idata$schema_repertoire is not NULL), the repertoire summaries will be recalculated based on the filtered data using agg_repertoires(). If FALSE, or if no repertoire schema exists, the returned ImmunData object will not contain repertoire summaries ($repertoires will be NULL).
.by	Not used.
.preserve	Not used.
barcodes	For filter_barcodes, a vector of cell identifiers (barcodes) to keep. Can be character, integer, or numeric.
receptors	For filter_receptors, a vector of receptor identifiers to keep. Can be character, integer, or numeric.

Details

For filter:

- User-provided dplyr-style filters (...) are applied *before* any sequence-based filtering defined in seq_options.
- Sequence filtering compares values in the query_col of the annotations table against the provided patterns.
- Supported sequence matching methods are:
  - "exact": Keeps rows where query_col exactly matches any of the patterns.
  - "regex": Keeps rows where query_col matches any of the regular expressions in patterns.
  - "lev" (Levenshtein distance): Keeps rows where the edit distance between query_col and any pattern is less than or equal to max_dist.
  - "hamm" (Hamming distance): Keeps rows where the Hamming distance (for equal length strings) between query_col and any pattern is less than or equal to max_dist.
- The filtering operations act on the $annotations table. A new ImmunData object is created containing only the rows (and corresponding receptors) that pass the filter(s).
- If keep_repertoires = TRUE (and repertoire data exists in the input), the repertoire-level summaries ($repertoires table) are recalculated based on the filtered annotations. Otherwise, the $repertoires table in the output will be NULL.

For filter_barcodes and filter_receptors:

- These functions provide a simpler interface for common filtering tasks based on cell barcodes or receptor IDs, respectively. They use efficient semi_join operations internally.

Value

A new ImmunData object containing only the filtered annotations (and potentially recalculated repertoire summaries). The schema remains the same.

See Also

make_seq_options(), dplyr::filter(), agg_repertoires(), ImmunData

Examples

```
# Basic setup (assuming idata_test is a valid ImmunData object)
# print(idata_test)

# --- filter examples ---
## Not run:
# Example 1: dplyr-style filtering on annotations
filtered_heavy <- filter(idata_test, chain == "IGH")
print(filtered_heavy)

# Example 2: Exact sequence matching on CDR3 amino acid sequence
cdr3_patterns <- c("CARGLGLVFYGMDVW", "CARDNRGAVAGVFGEAFYW")
seq_opts_exact <- make_seq_options(query_col = "CDR3_aa", patterns = cdr3_patterns)
filtered_exact_cdr3 <- filter(idata_test, seq_options = seq_opts_exact)
print(filtered_exact_cdr3)

# Example 3: Combining dplyr-style and fuzzy sequence matching (Levenshtein)
```

```
seq_opts_lev <- make_seq_options(
  query_col = "CDR3_aa",
  patterns = "CARGLGLVFYGMDVW",
  method = "lev",
  max_dist = 1
)
filtered_combined <- filter(idata_test,
  chain == "IGH",
  C_gene == "IGHG1",
  seq_options = seq_opts_lev
)
print(filtered_combined)

# Example 4: Regex matching on V gene
v_gene_pattern <- "^IGHV[13]-" # Keep only IGHV1 or IGHV3 families
seq_opts_regex <- make_seq_options(
  query_col = "V_gene",
  patterns = v_gene_pattern,
  method = "regex"
)
filtered_regex_v <- filter(idata_test, seq_options = seq_opts_regex)
print(filtered_regex_v)

# Example 5: Filtering without recalculating repertoires
filtered_no_rep <- filter(idata_test, chain == "IGK", keep_repertoires = FALSE)
print(filtered_no_rep) # $repertoires should be NULL

## End(Not run)

# --- filter_barcodes example ---
## Not run:
# Assuming 'cell1_barcode' and 'cell5_barcode' exist in idata_test$annotations$cell_id
specific_barcodes <- c("cell1_barcode", "cell5_barcode")
filtered_cells <- filter_barcodes(idata_test, barcodes = specific_barcodes)
print(filtered_cells)

## End(Not run)

# --- filter_receptors example ---
## Not run:
# Assuming receptor IDs 101 and 205 exist in idata_test$annotations$receptor_id
specific_receptors <- c(101, 205) # Or character IDs if applicable
filtered_recs <- filter_receptors(idata_test, receptors = specific_receptors)
print(filtered_recs)

## End(Not run)
```

---

from_immunarch *Convert an immunarch Object into an ImmunData Dataset*

Description

The `from_immunarch()` function takes an **immunarch** object (as returned by `immunarch::repLoad()`), writes each repertoire to a TSV file with an added `filename` column in a specified folder, and then imports those files into an **ImmunData** object via `read_repertoires()`.

Usage

```
from_immunarch(
  imm,
  output_folder,
  schema = c("CDR3.aa", "V.name"),
  temp_folder = file.path(tempdir(), "temp_folder")
)
```

Arguments

`imm`	A list returned by `immunarch::repLoad()`, typically containing: • `data`: a named list of `data.frames`, one per repertoire. • `meta`: (optional) a `data.frame` of sample metadata.
`output_folder`	Path to the output directory where the resulting ImmunData Parquet files will be stored. This directory will be created if it does not already exist.
`schema`	Character vector of column names that together define unique receptors (for example, `c("CDR3.aa", "V.name", "J.name")`).
`temp_folder`	Path to a directory where intermediate TSV files will be written. Defaults to `file.path(tempdir(), "temp_folder")`.

Value

An ImmunData object containing all repertoires from the input immunarch object, with data saved under `output_folder`.

See Also

`read_repertoires()`, `read_immundata()`, ImmunData

Examples

```
## Not run:
imm <- immunarch::repLoad("/path/to/your/files")
idata <- from_immunarch(imm,
  schema = c("CDR3.aa", "V.name"),
  temp_folder = tempdir(),
  output_folder = "/path/to/immundata_out"
)

## End(Not run)
```

imd_schema *Get Immundata internal schema field names*

Description

Returns the standardized field names used across Immundata objects and processing functions, as defined in `IMD_GLOBALS$schema`. These include column names for cell ids or barcodes, receptors, repertoires, and related metadata.

Usage

```
imd_schema(key = NULL)

imd_schema_sym(key = NULL)

imd_meta_schema()

imd_files()

imd_rename_cols(format = "default")

imd_drop_cols(format = "airr")

imd_repertoire_schema(format = "airr")

imd_receptor_features(schema)

imd_receptor_chains(schema)
```

Arguments

`key`	Character which field to return.
`format`	Character what format to load - "airr" or "10x".
`schema`	Receptor schema from `make_receptor_schema()`.

ImmunData *ImmunData: A Unified Structure for Immune Receptor Repertoire Data*

Description

`ImmunData` is an abstract R6 class for managing and transforming immune receptor repertoire data. It supports flexible backends (e.g., Arrow, DuckDB, dbplyr) and lazy evaluation, and provides tools for filtering, aggregation, and receptor-to-repertoire mapping.

Public fields

`schema_receptor` A named list describing how to interpret receptor-level data. This includes the fields used for aggregation (e.g., `CDR3`, `V_gene`, `J_gene`), and optionally unique identifiers for each receptor row. Used to ensure consistency across processing steps.

`schema_repertoire` A named list defining how barcodes or annotations should be grouped into repertoires. This may include sample-level metadata (e.g., `sample_id`, `donor_id`) used to define unique repertoires.

Active bindings

`receptors` Accessor for the dynamically-created table with receptors.

`annotations` Accessor for the annotation-level table (`.annotations`).

`repertoires` Get a table of repertoires and their basic statistics.

`metadata` Get a table of repertoires without their basic statistics.

Methods

Public methods:

- `ImmunData$new()`
- `ImmunData$clone()`

Method `new()`**:** Creates a new `ImmunData` object. This constructor expects receptor-level and barcode-level data, along with a receptor schema defining aggregation and identity fields.

Usage:

```
ImmunData$new(schema, annotations, repertoires = NULL)
```

Arguments:

`schema` A character vector specifying the receptor schema (e.g., aggregate fields, ID columns).

`annotations` A cell/barcode-level dataset mapping barcodes to receptor rows.

`repertoires` A repertoire table, created inside the body of agg_repertoires.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ImmunData$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

`read_repertoires()`, `read_immundata()`

make_default_preprocessing *Preprocessing and postprocessing of input immune repertoire files*

Description

Preprocessing and postprocessing of input immune repertoire files

Usage

```
make_default_preprocessing(format = c("airr", "10x"))

make_default_postprocessing()

make_exclude_columns(cols = imd_drop_cols("airr"))

make_productive_filter(col_name = c("productive"), truthy = TRUE)

make_barcode_prefix(prefix_col = "Prefix")
```

Arguments

format	For `make_default_preprocessing()`, a character string specifying the input data format. Currently supports `"airr"` (default) or `"10x"`. This determines the default set of columns to exclude and the values considered "productive".
cols	For `make_exclude_columns()`, a character vector of column names to be removed from the dataset. Defaults to `imd_drop_cols("airr")`. If empty, the returned function will not remove any columns.
col_name	For `make_productive_filter()`, a character vector of potential column names that indicate sequence productivity (e.g., `"productive"`). The first matching column found in the dataset will be used.
truthy	For `make_productive_filter()`, a value or vector of values that signify a productive sequence in the `col_name` column. Can be a logical `TRUE` (default for "airr" format) or a character vector of strings (e.g., `c("true", "TRUE", "True", "t", "T", "1")` for "10x" format).
prefix_col	For `make_barcode_prefix()`, the name of the column in the dataset that contains the prefix string to be added to each cell barcode. Defaults to `"Prefix"`. The barcode column itself is identified internally via `imd_schema("barcode")`.

Details

This collection of "maker" functions generates common preprocessing and postprocessing function steps tailored for immune repertoire data. Each `make_*` function returns a new function that can then be applied to a dataset.

These functions are designed to be flexible components in constructing custom data processing workflows.

The functions generated by these factories typically expect a `dataset` (e.g., a `duckplyr` with annotations) as their first argument and may accept additional arguments via `...` (though often unused in the predefined steps).

- `make_default_preprocessing()` and `make_default_postprocessing()` assemble a list of such processing functions.
- The individual `make_exclude_columns()`, `make_productive_filter()`, and `make_barcode_prefix()` functions create specific transformation steps.

These steps are often used when reading data to standardize formats, filter unwanted records, or enrich information like cell barcodes. They are designed to gracefully handle cases where an operation is not applicable (e.g., a specified column is not found) by issuing a warning and returning the dataset unmodified.

Value

Each `make_*` function returns a *new function*. This returned function takes a `dataset` as its first argument and `...` for any additional arguments, and performs the specific processing step. `make_default_preprocessing()` and `make_default_postprocessing()` return a *named list* of such functions.

Functions

- `make_default_preprocessing()`: Creates a default list of preprocessing functions suitable for "airr" or "10x" formatted data. This typically includes steps to exclude unnecessary columns and filter for productive sequences.
- `make_default_postprocessing()`: Creates a default list of postprocessing functions, such as adding a prefix to cell barcodes.
- `make_exclude_columns()`: Creates a function that, when applied to a dataset, removes a specified set of columns.
- `make_productive_filter()`: Creates a function that filters a dataset to retain only rows where sequences are marked as productive, based on a specified column and set of "truthy" values.
- `make_barcode_prefix()`: Creates a function that prepends a prefix (sourced from a specified column in the dataset) to the cell barcodes.

See Also

`read_repertoires()`

`make_receptor_schema` *Create or validate a receptor schema object*

Description

Helper functions for defining and validating the schema used by agg_receptors() to identify unique receptors.

make_receptor_schema() creates a schema list object. assert_receptor_schema() checks if an object is a valid schema list and throws an error if not. test_receptor_schema() checks if an object is a valid schema list or a character vector (which agg_receptors can also accept) and returns TRUE or FALSE.

Usage

```
make_receptor_schema(features, chains = NULL)

assert_receptor_schema(schema)

test_receptor_schema(schema)
```

Arguments

features	Character vector. Column names defining the features of a single receptor chain (e.g., V gene, J gene, CDR3 sequence).
chains	Optional character vector (max length 2). Locus names (e.g., "TRA", "TRB") to filter by or pair. If NULL or length 1, only filtering occurs. If length 2, pairing logic is enabled in agg_receptors(). Default: NULL.
schema	An object to test or assert as a valid schema. Can be a list created by make_receptor_schema or a character vector (for test_receptor_schema).

Value

make_receptor_schema returns a list with elements features and chains. assert_receptor_schema returns TRUE invisibly if valid, or stops execution. test_receptor_schema returns TRUE or FALSE.

---

make_seq_options *Build a* seq_options *list for sequence-based receptor filtering*

Description

A convenience wrapper that validates the common arguments for filter_receptors() and returns them in the required list form.

Usage

```
make_seq_options(
  query_col,
  patterns,
  method = c("exact", "lev", "hamm", "regex"),
  max_dist = NA,
  name_type = c("index", "pattern")
)
```

Arguments

query_col	Character(1). Name of the receptor column to compare (e.g. "cdr3_aa").
patterns	Character vector of sequences or regular expressions to search for.
method	One of "exact", "regex", "lev" (Levenshtein), or "hamm" (Hamming). Defaults to "exact".
max_dist	Numeric distance threshold for "lev" / "hamm" filtering. Use NA (default) to keep all rows after annotation.
name_type	Passed straight to annotate_tbl_distance(); either "index" (default) or "pattern".

Value

A named list suitable for the seq_options argument of filter_receptors().

See Also

filter_receptors(), annotate_receptors()

---

mutate_immundata *Modify or Add Columns to ImmunData Annotations*

Description

Applies transformations to the $annotations table within an ImmunData object, similar to dplyr::mutate. It allows adding new columns or modifying existing non-schema columns using standard dplyr expressions. Additionally, it can add new columns based on sequence comparisons (exact match, regular expression matching, or distance calculation) against specified patterns.

Usage

```
mutate_immundata(idata, ..., seq_options = NULL)

## S3 method for class 'ImmunData'
mutate(.data, ..., seq_options = NULL)
```

Arguments

idata, .data	An ImmunData object.
...	dplyr::mutate-style named expressions (e.g., new_col = existing_col * 2, category = ifelse(value > 10, "high", "low")). These are applied first. **Important**: You cannot use names for new or modified columns that conflict with the core ImmunData schema columns (retrieved via imd_schema()).
seq_options	Optional named list specifying sequence-based annotation options. Use make_seq_options() for convenient creation. See filter_immundata documentation (?filter_immundata) or the details section here for the list structure (query_col, patterns, method, name_type). max_dist is ignored for mutation. If NULL (the default), no sequence-based columns are added.

Details

The function operates in two main steps:

1. **Standard Mutations (...)**: Applies the standard `dplyr::mutate`-style expressions provided in `...` to the `$annotations` table. You can create new columns or modify existing ones, but you *cannot* modify columns defined in the core `ImmunData` schema (e.g., `receptor_id`, `cell_id`). An error will occur if you attempt to do so.
2. **Sequence-based Annotations (`seq_options`)**: If `seq_options` is provided, the function calculates sequence similarities or distances and adds corresponding new columns to the `$annotations` table.
   - `method = "exact"`: Adds boolean columns (TRUE/FALSE) indicating whether the `query_col` value exactly matches each `pattern`. Column names are generated using a prefix (e.g., `sim_exact_`) and the pattern or its index.
   - `method = "regex"`: Uses `annotate_tbl_regex` to add columns indicating matches for each regular expression pattern against the `query_col`. The exact nature of the added columns depends on `annotate_tbl_regex` (e.g., boolean flags or captured groups).
   - `method = "lev"` or `method = "hamm"`: Uses `annotate_tbl_distance` to calculate Levenshtein or Hamming distances between the `query_col` and each `pattern`, adding columns containing these numeric distances. `max_dist` is ignored in this context (internally treated as `NA`) as all distances are calculated and added, not used for filtering.
   - The naming of the new sequence-based columns depends on the `name_type` option within `seq_options` and internal helper functions like `make_pattern_columns`. Prefixes like `sim_exact_`, `sim_regex_`, `dist_lev_`, `dist_hamm_` are typically used based on the schema.

The `$repertoires` table, if present in the input `idata`, is copied to the output object without modification. This function only affects the `$annotations` table.

Value

A *new* `ImmunData` object with the `$annotations` table modified according to the provided expressions and `seq_options`. The `$repertoires` table (if present) is carried over unchanged from the input `idata`.

See Also

`dplyr::mutate()`, `make_seq_options()`, `filter_immundata()`, ImmunData, `vignette("immundata-classes", package = "immunarch")` (replace with actual package name if different)

Examples

```
# Basic setup (assuming idata_test is a valid ImmunData object)
# print(idata_test)

## Not run:
# Example 1: Add a simple derived column
idata_mut1 <- mutate(idata_test, V_family = substr(V_gene, 1, 5))
print(idata_mut1$annotations)

# Example 2: Add multiple columns and modify one (if 'custom_score' exists)
```

```
# Note: Avoid modifying core schema columns like 'V_gene' itself.
idata_mut2 <- mutate(idata_test,
  V_basic = gsub("-.*", "", V_gene),
  J_len = nchar(J_gene),
  custom_score = custom_score * 1.1
) # Fails if custom_score doesn't exist
print(idata_mut2$annotations)

# Example 3: Add boolean columns for exact CDR3 matches
cdr3_patterns <- c("CARGLGLVFYGMDVW", "CARDNRGAVAGVFGEAFYW")
seq_opts_exact <- make_seq_options(
  query_col = "CDR3_aa",
  patterns = cdr3_patterns,
  method = "exact",
  name_type = "pattern"
) # Name cols by pattern
idata_mut_exact <- mutate(idata_test, seq_options = seq_opts_exact)
# Look for new columns like 'sim_exact_CARGLGLVFYGMDVW'
print(idata_mut_exact$annotations)

# Example 4: Add Levenshtein distance columns for a CDR3 pattern
seq_opts_lev <- make_seq_options(
  query_col = "CDR3_aa",
  patterns = "CARGLGLVFYGMDVW",
  method = "lev",
  name_type = "index"
) # Name col like 'dist_lev_1'
idata_mut_lev <- mutate(idata_test, seq_options = seq_opts_lev)
# Look for new column 'dist_lev_1' (or similar based on schema)
print(idata_mut_lev$annotations)

# Example 5: Combine standard mutation and sequence annotation
seq_opts_regex <- make_seq_options(
  query_col = "V_gene",
  patterns = c(ighv1 = "^IGHV1-", ighv3 = "^IGHV3-"),
  method = "regex",
  name_type = "pattern"
)
idata_mut_combo <- mutate(idata_test,
  chain_upper = toupper(chain),
  seq_options = seq_opts_regex
)
# Look for 'chain_upper' and regex match columns (e.g., 'sim_regex_ighv1')
print(idata_mut_combo)

## End(Not run)
```

---

read_immundata *Load a saved ImmunData from disk*

Description

Reconstructs an ImmunData object from files previously saved to a directory by write_immundata() or the internal saving step of read_repertoires(). It reads the annotations.parquet file for the main data and metadata.json to retrieve the necessary receptor and repertoire schemas.

Usage

```
read_immundata(path, prudence = "stingy", verbose = TRUE)
```

Arguments

path	Character(1). Path to the **directory** containing the saved ImmunData files (annotations.parquet and metadata.json).
prudence	Character(1). Controls strictness of type inference when reading the Parquet file, passed to duckplyr::read_parquet_duckdb(). Default "stingy" likely implies stricter type checking or safer inference.
verbose	Logical(1). If TRUE (default), prints informative messages using cli during loading. Set to FALSE for quiet operation.

Details

This function expects a directory structure created by write_immundata(), containing at least:

- annotations.parquet: The main annotation data table.
- metadata.json: Contains package version, receptor schema, and optionally repertoire schema.

The loading process involves:

1. Checking that the specified path is a directory and contains the required annotations.parquet and metadata.json files.
2. Reading metadata.json using jsonlite::read_json().
3. Reading annotations.parquet using duckplyr::read_parquet_duckdb() with the specified prudence level.
4. Extracting the receptor_schema and repertoire_schema from the loaded metadata.
5. Instantiating a new ImmunData object using the loaded annotations data and the receptor_schema.
6. If a non-empty repertoire_schema was found in the metadata, it calls agg_repertoires() on the newly created object to recalculate and attach repertoire-level information based on that schema.

Value

A new ImmunData object reconstructed from the saved files. If repertoire information was saved, it will be recalculated and included.

See Also

write_immundata() for saving ImmunData objects, read_repertoires() for the primary data loading pipeline, ImmunData class, agg_repertoires() for repertoire definition.

Examples

```
## Not run:
# Assume 'my_idata' is an ImmunData object created previously
# my_idata <- read_repertoires(...)

# Define a temporary directory for saving
save_dir <- tempfile("saved_immundata_")

# Save the ImmunData object
write_immundata(my_idata, save_dir)

# --- Later, in a new session or script ---

# Load the ImmunData object back from the directory
loaded_idata <- read_immundata(save_dir)

# Verify the loaded object
print(loaded_idata)
# compare_methods(my_idata$annotations, loaded_idata$annotations) # If available

# Clean up
unlink(save_dir, recursive = TRUE)

## End(Not run)
```

`read_metadata` *Load and Validate Metadata Table for Immune Repertoire Files*

Description

This function loads a metadata table from either a file path or a data frame, validates the presence of a column with repertoire file paths, and converts all file paths to absolute paths. It is used to support flexible pipelines for loading bulk or single-cell immune repertoire data across samples.

If the input is a file path, the function attempts to read it with `readr::read_delim`. If the input is a data frame, it checks whether file paths are absolute; relative paths are only allowed when metadata is loaded from a file.

It warns the user if many of the files listed in the metadata table are missing, and stops execution if none of the files exist.

The column with file paths is normalized and renamed to match the internal filename schema.

Usage

```
read_metadata(metadata, filename_col = "File", delim = "\t", ...)
```

Arguments

metadata	A metadata table. Can be either: • a data frame with metadata, • or a path to a text/TSV/CSV file that can be read with readr::read_delim.
filename_col	A string specifying the name of the column in the metadata table that contains paths to repertoire files. Defaults to "File".
delim	Delimiter used to read the metadata file (if a path is provided). Defaults to "\t".
...	Additional arguments passed to readr::read_delim() when reading metadata from a file.

Value

A validated and updated metadata data frame with absolute file paths, and an additional column renamed according to IMD_GLOBALS$schema$filename.

---

read_repertoires *Read and process immune repertoire files to immundata*

Description

This is the main function for reading immune repertoire data into the immundata framework. It reads one or more repertoire files (AIRR TSV, 10X CSV, Parquet), performs optional preprocessing and column renaming, aggregates sequences into receptors based on a provided schema, optionally joins external metadata, performs optional postprocessing, and returns an ImmunData object.

The function handles different data types (bulk, single-cell) based on the presence of barcode_col and count_col. For efficiency with large datasets, it processes the data and saves intermediate results (annotations) as a Parquet file before loading them back into the final ImmunData object.

Usage

```
read_repertoires(
  path,
  schema,
  metadata = NULL,
  barcode_col = NULL,
  count_col = NULL,
  locus_col = NULL,
  umi_col = NULL,
  preprocess = make_default_preprocessing(),
  postprocess = make_default_postprocessing(),
  rename_columns = imd_rename_cols("10x"),
  enforce_schema = TRUE,
  metadata_file_col = "File",
  output_folder = NULL,
  repertoire_schema = NULL
)
```

Arguments

path	Character vector. Path(s) to input repertoire files (e.g., "/path/to/data/*.tsv.gz"). Supports glob patterns via Sys.glob(). Files can be Parquet, CSV, TSV, or gzipped versions thereof. All files must be of the same type. Alternatively, pass the special string "<metadata>" to read file paths from the metadata table (see metadata and metadata_file_col params).
schema	Defines how unique receptors are identified. Can be: • A character vector of column names (e.g., c("v_call", "j_call", "junction_aa")). • A schema object created by make_receptor_schema(), allowing specification of chains for pairing (e.g., make_receptor_schema(features = c("v_call", "junction_aa"), chains = c("TRA", "TRB"))).
metadata	Optional. A data frame containing metadata to be joined with the repertoire data, read by read_metadata() function. If path = "<metadata>", this table *must* be provided and contain the file paths column specified by metadata_file_col. Default: NULL.
barcode_col	Character(1). Name of the column containing cell barcodes or other unique cell/clone identifiers for single-cell data. Triggers single-cell processing logic in agg_receptors(). Default: NULL.
count_col	Character(1). Name of the column containing UMI counts or frequency counts for bulk sequencing data. Triggers bulk processing logic in agg_receptors(). Default: NULL. Cannot be specified if barcode_col is also specified.
locus_col	Character(1). Name of the column specifying the receptor chain locus (e.g., "TRA", "TRB", "IGH", "IGK", "IGL"). Required if schema specifies chains for pairing. Default: NULL.
umi_col	Character(1). Name of the column containing UMI counts for single-cell data. Used during paired-chain processing to select the most abundant chain per barcode per locus. Default: NULL.
preprocess	List. A named list of functions to apply sequentially to the raw data *before* receptor aggregation. Each function should accept a data frame (or duckplyr_df) as its first argument. See make_default_preprocessing() for examples. Default: make_default_preprocessing(). Set to NULL or list() to disable.
postprocess	List. A named list of functions to apply sequentially to the annotation data *after* receptor aggregation and metadata joining. Each function should accept a data frame (or duckplyr_df) as its first argument. See make_default_postprocessing() for examples. Default: make_default_postprocessing(). Set to NULL or list() to disable.
rename_columns	Named character vector. Optional mapping to rename columns in the input files using dplyr::rename() syntax (e.g., c(new_name = "old_name", barcode = "cell_id")). Renaming happens *before* preprocessing and schema application. See imd_rename_cols() for presets. Default: imd_rename_cols("10x").
enforce_schema	Logical(1). If TRUE (default), reading multiple files requires them to have the exact same columns and types. If FALSE, columns are unioned across files (potentially slower, requires more memory). Default: TRUE.

`metadata_file_col`
: Character(1). The name of the column in the `metadata` table that contains the full paths to the repertoire files. Only used when `path = "<metadata>"`. Default: `"File"`.

`output_folder`
: Character(1). Path to a directory where intermediate processed annotation data will be saved as `annotations.parquet` and `metadata.json`. If `NULL` (default), a folder named `immundata-<basename_without_ext>` is created in the same directory as the first input file specified in `path`. The final `ImmunData` object reads from these saved files. Default: `NULL`.

`repertoire_schema`
: Character vector or Function. Defines columns used to group annotations into distinct repertoires (e.g., by sample or donor). If provided, `agg_repertoires()` is called after loading to add repertoire-level summaries and metrics. Default: `NULL`.

Details

The function executes the following steps:

1. Validates inputs.
2. Determines the list of input files based on `path` and `metadata`. Checks file extensions.
3. Reads data using `duckplyr` (`read_parquet_duckdb` or `read_csv_duckdb`). Handles `.gz`.
4. Applies column renaming if `rename_columns` is provided.
5. Applies preprocessing steps sequentially if `preprocess` is provided.
6. Aggregates sequences into receptors using `agg_receptors()`, based on `schema`, `barcode_col`, `count_col`, `locus_col`, and `umi_col`. This creates the core annotation table.
7. Joins the `metadata` table if provided.
8. Applies postprocessing steps sequentially if `postprocess` is provided.
9. Creates a temporary `ImmunData` object in memory.
10. Determines the `output_folder` path.
11. Saves the processed annotation table and metadata using `write_immundata()` to the `output_folder`.
12. Loads the data back from the saved Parquet files using `read_immundata()` to create the final `ImmunData` object. This ensures the returned object is backed by efficient storage.
13. If `repertoire_schema` is provided, calls `agg_repertoires()` on the loaded object to define and summarize repertoires.
14. Returns the final `ImmunData` object.

Value

An `ImmunData` object containing the processed receptor annotations. If `repertoire_schema` was provided, the object will also contain repertoire definitions and summaries calculated by `agg_repertoires()`.

See Also

ImmunData, `read_immundata()`, `write_immundata()`, `read_metadata()`, `agg_receptors()`, `agg_repertoires()`, `make_receptor_schema()`, `make_default_preprocessing()`, `make_default_postprocessing()`

Examples

```
## Not run:
#
# Example 1: single-chain, one file
#
# Read a single AIRR TSV file, defining receptors by V/J/CDR3_aa
# Assume "my_sample.tsv" exists and follows AIRR format

# Create a dummy file for illustration
airr_data <- data.frame(
  sequence_id = paste0("seq", 1:5),
  v_call = c("TRBV1", "TRBV1", "TRBV2", "TRBV1", "TRBV3"),
  j_call = c("TRBJ1", "TRBJ1", "TRBJ2", "TRBJ1", "TRBJ1"),
  junction_aa = c("CASSL...", "CASSL...", "CASSD...", "CASSL...", "CASSF..."),
  productive = c(TRUE, TRUE, TRUE, FALSE, TRUE),
  locus = c("TRB", "TRB", "TRB", "TRB", "TRB")
)
readr::write_tsv(airr_data, "my_sample.tsv")

# Define receptor schema
receptor_def <- c("v_call", "j_call", "junction_aa")

# Specify output folder
out_dir <- tempfile("immundata_output_")

# Read the data (disabling default preprocessing for this simple example)
idata <- read_repertoires(
  path = "my_sample.tsv",
  schema = receptor_def,
  output_folder = out_dir,
  preprocess = NULL, # Disable default productive filter for demo
  postprocess = NULL # Disable default barcode prefixing
)

print(idata)
print(idata$annotations)

#
# Example 2: single-chain, multiple files
#
# Read multiple files using metadata
# Create dummy files and metadata
readr::write_tsv(airr_data[1:2, ], "sample1.tsv")
readr::write_tsv(airr_data[3:5, ], "sample2.tsv")
meta <- data.frame(
  SampleID = c("S1", "S2"),
  Tissue = c("PBMC", "Tumor"),
  FilePath = c(normalizePath("sample1.tsv"), normalizePath("sample2.tsv"))
)
readr::write_tsv(meta, "metadata.tsv")

idata_multi <- read_repertoires(
```

```
  path = "<metadata>",
  metadata = meta,
  metadata_file_col = "FilePath",
  schema = receptor_def,
  repertoire_schema = "SampleID", # Aggregate by SampleID
  output_folder = tempfile("immundata_multi_"),
  preprocess = make_default_preprocessing("airr"), # Use default AIRR filters
  postprocess = NULL
)

print(idata_multi)
print(idata_multi$repertoires) # Check repertoire summary

# Clean up dummy files
file.remove("my_sample.tsv", "sample1.tsv", "sample2.tsv", "metadata.tsv")
unlink(out_dir, recursive = TRUE)
unlink(attr(idata_multi, "output_folder"), recursive = TRUE) # Get path used by function

## End(Not run)
```

`write_immundata` *Save ImmunData to disk*

Description

Serializes the essential components of an `ImmunData` object to disk for efficient storage and later retrieval. It saves the core annotation data (`idata$annotations`) as a compressed Parquet file and accompanying metadata (including receptor/repertoire schemas and package version) as a JSON file within a specified directory.

Usage

```
write_immundata(idata, output_folder)
```

Arguments

`idata`	The `ImmunData` object to save. Must be an R6 object of class `ImmunData` containing at least the `$annotations` table and schema information (`$schema_receptor`, optionally `$schema_repertoire`).
`output_folder`	Character(1). Path to the directory where the output files will be written. If the directory does not exist, it will be created recursively.

Details

The function performs the following actions:

1. Validates the input `idata` object and `output_folder` path.
2. Creates the `output_folder` if it doesn't exist.

3. Constructs a list containing metadata: `immundata` package version, receptor schema (`idata$schema_receptor`), and repertoire schema (`idata$schema_repertoire`).
4. Writes the metadata list to `metadata.json` within `output_folder`.
5. Writes the `idata$annotations` table (a `duckplyr_df` or similar) to `annotations.parquet` within `output_folder`. Uses Zstandard compression (`compression = "zstd"`, `compression_level = 9`) for a good balance between file size and read/write speed.
6. Uses internal helper `imd_files()` to determine the standard filenames (`metadata.json`, `annotations.parquet`).

The receptor data itself (if stored separately in future versions) is not saved by this function; only the annotations linking to receptors are saved, along with the schema needed to reconstruct/interpret them.

Value

Invisibly returns the input `idata` object, saved to disk. In other words, this allows you to create snapshots of the data in the `output_folder`. Mind that by saving the object, you execute all the stored computations, so this operations can take longer than expected. Read more about snapshots on our website in the "Concept" section.

See Also

`read_immundata()` for loading the saved data, `read_repertoires()` which uses this function internally, ImmunData class definition.

Examples

```
## Not run:
# Assume 'my_idata' is an ImmunData object created previously
# my_idata <- read_repertoires(...)

# Define an output directory
save_dir <- tempfile("saved_immundata_")

# Save the ImmunData object
write_immundata(my_idata, save_dir)

# Check the created files
list.files(save_dir) # Should show "annotations.parquet" and "metadata.json"

# Clean up
unlink(save_dir, recursive = TRUE)

## End(Not run)
```

Index

∗ **Annotation**
  annotate_immundata, 6
  annotate_seurat, 9
∗ **aggregation**
  agg_receptors, 2
  agg_repertoires, 4
∗ **annotation**
  annotate_immundata, 6
∗ **core_immundata**
  ImmunData, 15
∗ **filtering**
  filter_immundata, 10
∗ **ingestion**
  from_immunarch, 13
  read_immundata, 22
  read_metadata, 24
  read_repertoires, 25
  write_immundata, 29
∗ **mutation**
  mutate_immundata, 20
∗ **operations**
  count.ImmunData, 10
∗ **processing**
  make_default_preprocessing, 17
∗ **schema**
  imd_schema, 15
∗ **utils**
  make_receptor_schema, 18
  make_seq_options, 19

agg_receptors, 2
agg_receptors(), *19*, *26, 27*
agg_repertoires, 4, *16*
agg_repertoires(), *11, 12*, *23*, *27*
annotate *(*annotate_immundata*)*, 6
annotate_barcodes *(*annotate_immundata*)*, 6
annotate_chains *(*annotate_immundata*)*, 6
annotate_immundata, 6
annotate_receptors *(*annotate_immundata*)*, 6
annotate_receptors(), *20*
annotate_seurat, 9
assert_receptor_schema *(*make_receptor_schema*)*, 18

count.ImmunData, 10

dplyr::filter(), *12*
dplyr::mutate(), *21*

filter.ImmunData *(*filter_immundata*)*, 10
filter_barcodes *(*filter_immundata*)*, 10
filter_immundata, 10
filter_immundata(), *21*
filter_receptors *(*filter_immundata*)*, 10
filter_receptors(), *20*
from_immunarch, 13

imd_drop_cols *(*imd_schema*)*, 15
imd_files *(*imd_schema*)*, 15
imd_meta_schema *(*imd_schema*)*, 15
imd_receptor_chains *(*imd_schema*)*, 15
imd_receptor_features *(*imd_schema*)*, 15
imd_rename_cols *(*imd_schema*)*, 15
imd_rename_cols(), *26*
imd_repertoire_schema *(*imd_schema*)*, 15
imd_schema, 15
imd_schema_sym *(*imd_schema*)*, 15
ImmunData, *4, 5*, *9*, *12*, *14*, 15, *21*, *23*, *27*, *30*

make_barcode_prefix *(*make_default_preprocessing*)*, 17
make_default_postprocessing *(*make_default_preprocessing*)*, 17
make_default_postprocessing(), *26, 27*
make_default_preprocessing, 17
make_default_preprocessing(), *26, 27*

make_exclude_columns
(make_default_preprocessing), 17
make_productive_filter
(make_default_preprocessing), 17
make_receptor_schema, 18
make_receptor_schema(), *4*, *15*, *26*, *27*
make_seq_options, 19
make_seq_options(), *11*, *12*, *20*, *21*
mutate.ImmunData (mutate_immundata), 20
mutate_immundata, 20

read_immundata, 22
read_immundata(), *5*, *14*, *16*, *27*, *30*
read_metadata, 24
read_metadata(), *26*, *27*
read_repertoires, 25
read_repertoires(), *2*, *4*, *5*, *14*, *16*, *18*, *23*, *30*

SeuratObject::AddMetaData, *9*
Sys.glob(), *26*

test_receptor_schema
(make_receptor_schema), 18

write_immundata, 29
write_immundata(), *23*, *27*