Package ‘ClustIRR’

December 23, 2024

Type Package

Title Clustering of immune receptor repertoires

Version 1.4.0

Description ClustIRR analyzes repertoires of B- and T-cell receptors. It starts by identifying communities of immune receptors with similar specificities, based on the sequences of their complementarity-determining regions (CDRs). Next, it employs a Bayesian probabilistic models to quantify differential community occupancy (DCO) between repertoires, allowing the identification of expanding or contracting communities in response to e.g. infection or cancer treatment.

License GPL-3 + file LICENSE

LazyData false

Depends R (>= 4.4.0)

Imports blaster, future, future.apply, grDevices, igraph, methods, pwalign, Rcpp (>= 0.12.0), RcppParallel (>= 5.0.1), reshape2, rstan (>= 2.18.1), rstantools (>= 2.4.0), stats, stringdist, utils, visNetwork

Suggests BiocStyle, knitr, testthat, ggplot2, ggrepel, patchwork

Encoding UTF-8

NeedsCompilation no

biocViews Clustering, ImmunoOncology, SingleCell, Software, Classification

RoxygenNote 7.2.3

VignetteBuilder knitr

URL https://github.com/snaketron/ClustIRR

BugReports https://github.com/snaketron/ClustIRR/issues

Biarch true

LinkingTo BH (>= 1.66.0), Rcpp (>= 0.12.0), RcppEigen (>= 0.3.3.3.0), RcppParallel (>= 5.0.1), rstan (>= 2.18.1), StanHeaders (>= 2.18.0)

SystemRequirements GNU make

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

git_branch RELEASE_3_20

git_last_commit ef4afb4

git_last_commit_date 2024-10-29

Repository Bioconductor 3.20

Date/Publication 2024-12-23

Author Simo Kitanovski [aut, cre] (<https://orcid.org/0000-0003-2909-5376>),
Kai Wollek [aut] (<https://orcid.org/0009-0008-5941-9160>)

Maintainer Simo Kitanovski <simokitanovski@gmail.com>

Contents

CDR3ab — *Mock data set of complementarity determining region 3 (CDR3) sequences from the α and β chains of 10,000 T cell receptors*

Description

Mock data set containing amino acid sequences of paired CDR3s from the α and β chains of 10,000 T cell receptors. All CDR3 sequences were drawn from a larger set of CDR3β sequences from human naive CD8+ T cells.

Usage

```
data(CDR3ab)
```

Format

`data.frame` with 10,000 rows and 2 columns `CDR3a` and `CDR3b`.

Value

data(CDR3ab) loads the object CDR3ab, which is a data.frame with two columns and 10,000 rows.

Source

GLIPH version 2

Examples

```
data("CDR3ab")
```

`cluster_irr` *Clustering of immune receptor repertoires (IRRs)*

Description

`cluster_irr` computes similarities between immune receptors (IRs = T-cell and B-cell receptors) based on their CDR3 sequences.

Usage

```
cluster_irr(s,
            control = list(gmi = 0.7,
                           trim_flank_aa = 3,
                           db_dist = 0,
                           db_custom = NULL))
```

Arguments

`s` a data.frame with complementarity determining region 3 (CDR3) amino acid sequences observed in IRR clones (data.frame rows). The data.frame has the following columns (IR clone features):

- `sample`: name of the IRR (e.g. 'A')
- `clone_size`: cell count in the clone (=clonal expansion)
- `CDR3?`: amino acid CDR3 sequence. Replace '?' with the appropriate name of the IR chain (e.g. CDR3a for CDR3s from TCRα chain; or CDR3d for CDR3s from TCRδ chain. Meanwhile, if paired CDR3s from both chains are available, then you can provide both in separate columns e.g.:
  - *CDR3b* and *CDR3a* [for $\alpha\beta$ TCRs]
  - *CDR3g* and *CDR3d* [for $\gamma\delta$ TCRs]
  - *CDR3h* and *CDR3l* [for heavy/light chain BCRs]

`control` auxiliary parameters to control the algorithm's behavior. See the details below:

- `gmi`: the minimum sequence identity between a pair of CDR3 sequences for them to even be considered for alignment and scoring (default = 0.7; 70 percent identity).
- `trim_flank_aa`: how many amino acids should be trimmed from the flanks of all CDR3 sequences to isolate the **CDR3 cores**. `trim_flank_aa = 3` (default).
- `db_custom`: additional database (data.frame) which allows us to annotate CDR3 sequences from the input (`s`) with their cognate antigens. The structure of `db_custom` must be identical to that in `data(vdjdb, package = "ClustIRR")`. ClustIRR will use the internal databases if `db_custom`=NULL (default). Three databases (**data only from human CDR3**) are integrated in ClustIRR: `VDJdb`, `TCR3d` and `McPAS-TCR`.
- `db_dist`: we compute edit distances between CDR3 sequences from `s` and from a database (e.g. VDJdb). If a particular distance is smaller than or equal to `edit_dist` (default = 0), then we annotate the CDR3 from `s` with the specificity of the database CDR3 sequence.

Details

IRRs, such as T-cell receptor repertoires, are made up of T-cells which are distributed over T-cell clones. TCR clones with **identical** pairs of CDR3α and CDR3β sequences most likely recognize the same sets of antigens. Meanwhile, TCR clones with **similar** pairs of CDR3α and CDR3β sequences may also share common specificity. ClustIRR aims to quantify the similarity between pairs of TCR clones based on the similarities of their CDR3s sequences.

How to compute a similarity score between a pair of CDR3 sequences?

Pair of sequences, a and b, are aligned with the Needleman-Wunsch algorithm (BLOSUM62 substitution matrix is used for scoring). The output is an alignment score (ω). Identical or similar CDR3 sequence pairs get a large positive ω, and dissimilar CDR3 sequence pairs get a low (or even negative) ω.

To make sure that ω is comparable across pairs of CDR3s with different lengths, ClustIRR divides (normalizes) ω by the length of the longest CDR3 sequences in each pair:

$$\bar{\omega} = \frac{\omega}{\max(|a|, |b|)}$$

where $|a|$ and $|b|$ are the lengths of CDR3 sequences a and b; and $\bar{\omega}$ is the normalized alignment score.

The CDR3 **cores**, which represent the central parts of the CDR3 loop and tend to have high probability of making a contact with the antigen, are compared with the same procedure. ClustIRR constructs the CDR3 cores by trimming few residues (defined by `control$trim_flanks`) from either end of each CDR3 sequences. These are then aligned and scored based on the same algorithm, yielding for each pair of CDR3 cores a normalized alignment scores $\bar{\omega}_c$.

This strategy is computationally very expensive!

For large IRRs with $n > 10^6$ this algorithm requires significant computational resources. To mitigate this challenge, we employ a screening step in which dissimilar sequences pairs are flagged. In short, each CDR3 is used as a query in a **fast** protein-BLAST search as implemented in the R-package blaster, while the remaining CDR3s are considered as a database of amino acid sequences against which the query is compared. CDR3 sequences which share at least 70% sequence identity (user parameter `control$gmi`) with the query are selected, and only these are aligned with query CDR3. For the remaining CDR3 pairs we assume $\bar{\omega} = 0$.

Value

The output is an `S4` object of class `clust_irr`. This object contains two sublists:

- `clust`, list, contains clustering results for each IR chain. The results are stored as data.frame in separate sub-list named appropriately (e.g. CDR3a, CDR3b, CDR3g, etc.). Each row in the data.frames contains a pair of CDR3s.

  The remaining columns contain similarity scores for the complete CDR3 sequences (column `weight`) or their cores (column `cweight`). The columns `max_len` and `max_clen` store the length of the longer CDR3 sequence and core in the pair, and these used to normalize the scores `weight` and `cweight`: the normalized scores are shown in the columns `nweight` and `ncweight`
- `inputs`, list, contains all user provided inputs (see Arguments)

Examples

```
# load package input data
data("CDR3ab", package = "ClustIRR")
s <- data.frame(CDR3b = CDR3ab[1:100, "CDR3b"], sample = "A", clone_size = 1)
```

```
# run analysis
c <- cluster_irr(s = s)

# output class
class(c)

# output structure
str(c)

# inspect which CDR3bs are similar
knitr::kable(head(slot(c, "clust")$CDR3b))
```

`clust_irr-class` *clust_irr class*

Description

Objects of the class `clust_irr` are generated by the function `cluster_irr`. These objects are used to store the clustering results in a structured way, such that they may be used as inputs of other ClustIRR functions (e.g. get_graph, plot_graph, etc.).

The output is an `S4` object of class `clust_irr`. This object contains two sublists:

- `clust`, list, contains clustering results for each IR chain. The results are stored as data.frame in separate sub-list named appropriately (e.g. CDR3a, CDR3b, CDR3g, etc.). Each row in the data.frames contains a pair of CDR3s.

  The remaining columns contain similarity scores for the complete CDR3 sequences (column `weight`) or their cores (column `cweight`). The columns `max_len` and `max_clen` store the length of the longer CDR3 and CDR3 core sequence in the pair, and these used to normalize the scores `weight` and `cweight`: the normalized scores are shown in the columns `nweight` and `ncweight`
- `inputs`, list, contains all user provided inputs (see Arguments)

Arguments

`clust`	list, contains clustering results for each TCR/BCR chain. The results are stored in separate sub-list named appropriately (e.g. CDR3a, CDR3b, CDR3g, etc.)
`inputs`	list, contains all user provided inputs

Value

The output is an S4 object of class `clust_irr`

Accessors

To access the slots of `clust_irr` object we have two accessor functions. In the description below, `x` is a `clust_irr` object.

get_clustirr_clust `get_clustirr_clust(x)`: Extract the clustering results (slot clust)

get_clustirr_inputs `get_clustirr_inputs(x)`: Extract the processed inputs (slot inputs)

Examples

```
# load package input data
data("CDR3ab", package = "ClustIRR")
s <- data.frame(CDR3b = CDR3ab[1:100, "CDR3b"], sample = "A", clone_size = 1)

# run analysis
c <- cluster_irr(s = s)

# output class
class(c)

# output structure
str(c)

# inspect which CDR3bs are globally similar
knitr::kable(head(slot(c, "clust")$CDR3b))


# clust_irr S4 object generated 'manually' from the individual results
new_clust_irr <- new("clust_irr",
                     clust = slot(object = c, name = "clust"),
                     inputs = slot(object = c, name = "inputs"))

# we should get identical outputs
identical(x = new_clust_irr, y = c)
```

---

dco — *Model-based differential community occupancy (DCO) analysis*

Description

This algorithm takes as input a community matrix, and quantifies the relative enrichment/depletion of individual communities in each sample using a Bayesian hierarchical model.

Usage

```
dco(community_occupancy_matrix, mcmc_control)
```

Arguments

`community_occupancy_matrix`
: matrix, rows are communities, columns are repertoires, matrix entries are numbers of cells in each community and repertoire.

`mcmc_control`
: list, configurations for the Markov Chain Monte Carlo (MCMC) simulation.

- mcmc_warmup = 750; number of MCMC warmups
- mcmc_iter = 1500; number of MCMC iterations
- mcmc_chains = 4; number of MCMC chains
- mcmc_chains = 1; number of computer cores
- mcmc_algorithm = "NUTS"; which MCMC algorithm to use
- adapt_delta = 0.95; MCMC step size
- max_treedepth = 12; the max value, in exponents of 2, of what the binary tree size in NUTS should have.

Value

The output is a list with the folling elements:

- `fit` stan object, model fit
- `posterior_summary` nested list with data.frames, summary of model parameters, including their means, medians, 95% credible intervals, etc. Predicted observations (y_hat), which are useful for posterior predictive checks are also provided.
- `community_occupancy_matrix` matrix, rows are communities, columns are repertoires, matrix entries are numbers of cells in each community and repertoire.
- `mcmc_control` list, mcmc configuration inputs provided as list.

Examples

```
# load package input data
data("CDR3ab", package = "ClustIRR")
a <- data.frame(CDR3a = CDR3ab[1:500, "CDR3a"],
                CDR3b = CDR3ab[1:500, "CDR3b"],
                clone_size = 1,
                sample = "a")

b <- data.frame(CDR3a = CDR3ab[401:900, "CDR3a"],
                CDR3b = CDR3ab[401:900, "CDR3b"],
                clone_size = 1,
                sample = "b")
b$clone_size[1] <- 20

# run ClustIRR analysis
c <- c(cluster_irr(s = a), cluster_irr(s = b))

# get joint graph
jg <- get_joint_graph(clust_irrs = c)

# detect communities
gcd <- detect_communities(graph = jg$graph,
                          algorithm = "leiden",
                          resolution = 1,
                          weight = "ncweight",
                          metric = "average",
                          chains = c("CDR3a", "CDR3b"))

# look at outputs
names(gcd)

# look at the community matrix
head(gcd$community_occupancy_matrix)

# look at the community summary
head(gcd$community_summary)

# look at the node summary
head(gcd$node_summary)

# differential community occupancy analysis
```

```
dco <- dco(community_occupancy_matrix = gcd$community_occupancy_matrix)

names(dco)
```

---

detect_communities *Graph-based community detection (GCD)*

Description

Performs graph-based community detection to find densely connected groups of nodes in graph constructed by `get_graph` or `get_joint_graph`.

Usage

```
detect_communities(graph,
                   algorithm = "leiden",
                   resolution = 1,
                   weight = "ncweight",
                   metric = "average",
                   chains)
```

Arguments

`graph`	`igraph` object
`algorithm`	graph-based community detection (GCD) method: leiden (default) or louvain.
`resolution`	clustering resolution (default = 1) for the GCD.
`weight`	which edge weight metric (default = ncweight) should be used for GCD
`metric`	possible metrics: "average" (default), "strict" or "loose".
`chains`	which chains should be used for clustering? For instance: chains = "CDR3a"; or chains = CDR3b; or chains = c("CDR3a", "CDR3b").

Details

ClustIRR employs graph-based community detection (GCD) algorithms, such as Louvain or Leiden, to identify densely connected nodes. But first, we must decide how to compute a similarity between two nodes, i and j, (e.g. TCR clones) based on the similarity scores between their CDR3 sequences (computed in `clust_irr`) and use this metric as edge weight $\omega(i, j)$.

Scenario 1

If our IRR data data contains CDR3 sequences from only one chain, such as CDR3β, then $\omega(i, j)$ is defined as

$\omega(i, j) = \bar{\omega}^{\beta}$ or $\omega(i, j) = \bar{\omega}_c^{\beta}$

The user can decide among the two definitions by specifying

- `weight` = "ncweight" $\rightarrow \omega(i, j) = \bar{\omega}_c$ (default)
- `weight` = "nweight" $\rightarrow \omega(i, j) = \bar{\omega}$

Scenario 2

If our IRR data contains CDR3 sequences from both chains (paired data) To compute the similarity score between TCR clones, i and j, we compute the average alignment score (`metric=`**average**) from their CDR3α and CDR3β alignment scores (in the next, I will use TCR$\alpha\beta$ as an example, however, this approach can also be used to compare TCR$\gamma\delta$ or BCR*IgH-IgL* clones):

$$\omega(i,j) = \frac{\bar{\omega}^{\alpha} + \bar{\omega}^{\beta}}{2} \qquad \text{or} \qquad \omega(i,j) = \frac{\bar{\omega}_c^{\alpha} + \bar{\omega}_c^{\beta}}{2},$$

where $\bar{\omega}^{\alpha}$ and $\bar{\omega}^{\beta}$ are the alignment scores for the CDR3α and CDR3β sequences, respectively; and $\bar{\omega}_c^{\alpha}$ and $\bar{\omega}_c^{\beta}$ are the alignment scores for the CDR3α and CDR3β cores, respectively. Based on this metric, CDR3α and CDR3β contribute towards the overall similarity of the TCR clones with equal weights.

ClustIRR provides two additional metrics for computing similarity scores between TCR clones, including a `metric=`**strict**, which assigns high similarity score to a pair of TCR clones only if both of their CDR3α and CDR3β sequence pairs are similar

$$\omega(i,j) = \min(\bar{\omega}^{\alpha}, \bar{\omega}^{\beta}) \qquad \text{or} \qquad \omega(i,j) = \min(\bar{\omega}_c^{\alpha}, \bar{\omega}_c^{\beta}),$$

and a `metric=`**loose**, which assigns high similarity score to a pair of TCR clones if either of their CDR3α and CDR3β sequence pairs are similar

$$\omega(i,j) = \max(\bar{\omega}^{\alpha}, \bar{\omega}^{\beta}) \qquad \text{or} \qquad \omega(i,j) = \max(\bar{\omega}_c^{\alpha}, \bar{\omega}_c^{\beta}),$$

Value

The output is a list with the folling elements:

`community_occupancy_matrix`
: matrix, rows are communities, columns are repertoires, matrix entries are numbers of cells in each community and repertoire.

`community_summary`
: data.frame, rows are communities and their properties are provided as columns.

`node_summary`
: data.frame, rows are nodes (clones) and their properties are provided as columnscontains all user provided.

`graph`
: igraph object, processed graph object

`input_config`
: list, inputs provided as list.

Examples

```
# load package input data
data("CDR3ab", package = "ClustIRR")
a <- data.frame(CDR3a = CDR3ab[1:300, "CDR3a"],
                CDR3b = CDR3ab[1:300, "CDR3b"],
                clone_size = 1,
                sample = "a")

b <- data.frame(CDR3a = CDR3ab[201:400, "CDR3a"],
                CDR3b = CDR3ab[201:400, "CDR3b"],
                clone_size = 1,
                sample = "b")
b$clone_size[1] <- 20

# run ClustIRR analysis
c <- c(cluster_irr(s = a), cluster_irr(s = b))

# get joint graph
```

```
jg <- get_joint_graph(clust_irrs = c)

# detect communities
gcd <- detect_communities(graph = jg$graph,
                          algorithm = "leiden",
                          resolution = 1,
                          weight = "ncweight",
                          metric = "average",
                          chains = c("CDR3a", "CDR3b"))

# look at outputs
names(gcd)

# look at the community occupancymatrix
head(gcd$community_occupancy_matrix)

# look at the community summary
head(gcd$community_summary)

# look at the node summary
head(gcd$node_summary)
```

`get_graph` *Get* `igraph` *object from* `clust_irr` *object*

Description

Given a `clust_irr` object generated by the function `cluster_irr`, the function `get_graph` constructs an `igraph` object.

The graph nodes represent IR clones. Undirected edges are drawn between pairs of nodes, and the attributes of these edges are assigned based on the `clust_irr` outputs: $\bar{\omega}$, $\bar{\omega}_c$, etc.

Usage

```
get_graph(clust_irr)
```

Arguments

`clust_irr`	S4 object generated by the function `cluster_irr`

Value

The output is a list with the following elements. First, the list contains an `igraph` object. The graph nodes and edges contain attributes encoded in the `clust_irr` objects. Second, it contains a data.frame in which rows are clones (nodes) in the graph. Third, the list contains the logical variable `joint_graph`, which is set to `TRUE` if the graph is a joint graph generated by the function `get_joint_graph` and `FALSE` if the graph is not a joint graph generated by `get_graph`.

Examples

```
# load package input data
data("CDR3ab", package = "ClustIRR")
s <- data.frame(CDR3b = CDR3ab[1:100, "CDR3b"], sample = "A", clone_size = 1)

# run ClustIRR analysis
out <- cluster_irr(s = s)

# get graph
g <- get_graph(clust_irr = out)

names(g)
```

`get_joint_graph` *Create joint* `igraph` *object from multiple* `clust_irr` *objects*

Description

Given a vector of `clust_irr` objects, generated by the function `cluster_irr`, the function `get_joint_graph` performs the following steps:

1. runs the function `get_graph` on each `clust_irr` object

2. merges the nodes: if graph `a` and `b` have `|a|` and `|b|` nodes, then the joint graph has `|a|+|b|` nodes, regardless of whether exactly the same clone (vertex) is found in both graphs.

3. draws edges between nodes from the different graphs using the same algorithm for drawing edges between nodes within an IRR (see function `clust_irr`).

4. return a joint graph as `igraph` object

5. return a data.frame with all clones (graph nodes)

6. return a logical `joint_graph`=TRUE

Usage

```
get_joint_graph(clust_irrs, cores = 1)
```

Arguments

`clust_irrs`	A list of at least two S4 objects generated with the function `cluster_irr`
`cores`	number of computer cores to use (default = 1)

Value

The main output is an `igraph` object.

Examples

```
# load package input data
data("CDR3ab", package = "ClustIRR")
a <- data.frame(CDR3b = CDR3ab[1:100, "CDR3b"], sample = "a", clone_size = 1)
b <- data.frame(CDR3b = CDR3ab[1:100, "CDR3b"], sample = "b", clone_size = 1)

# run ClustIRR analysis
```

```
c <- c(cluster_irr(s = a), cluster_irr(s = b))

# get graph
g <- get_joint_graph(clust_irrs = c)

names(g)
```

---

mcpas *CDR3 sequences and their matching epitopes obtained from McPAS-TCR*

Description

data.frame with `CDR3a` and/or `CDR3b` sequences and their matching antigenic epitopes obtained from McPAS-TCR. The remaining CDR3 columns are set to NA. For data processing details see the script inst/script/get_mcpastcr.R

Usage

```
data(mcpas)
```

Format

`data.frame` with columns:

1. `CDR3a`: CDR3a amino acid sequence
2. `CDR3b`: CDR3b amino acid sequence
3. `CDR3g`: CDR3g amino acid sequence -> NA
4. `CDR3d`: CDR3d amino acid sequence -> NA
5. `CDR3h`: CDR3h amino acid sequence -> NA
6. `CDR3l`: CDR3l amino acid sequence -> NA
7. `CDR3_species`: CDR3 species (e.g. human, mouse, ...)
8. `Antigen_species`: antigen species
9. `Antigen_gene`: antigen gene
10. `Reference`: Reference (Pubmed ID)

Value

data(mcpas) loads the object McPAS-TCR

Source

McPAS-TCR, June 2024

Examples

```
data(mcpas)
```

plot_graph *Plot ClustIRR graph*

Description

This function visualizes a graph. The main input is g object created by the function `get_graph`.

Usage

```
plot_graph(g,
           select_by = "Ag_species",
           as_visnet = FALSE,
           show_singletons = TRUE,
           node_opacity = 1)
```

Arguments

`g`	Object returned by the functions `get_graph` or `get_joint_graph`
`as_visnet`	logical, if `as_visnet`=TRUE we plot an interactive graph with visNetwork. If `as_visnet`=FALSE, we plot a static graph with igraph.
`select_by`	character string, two values are possible: "Ag_species" or "Ag_gene". This only has an effect if `as_visnet` = TRUE, i.e. if the graph is interactive. It will allow the user to highligh clones (nodes) in the graph that are associated with a specific antigenic specie or gene. The mapping between CDR3 and antigens is extracted from databases, such as, VDJdb, McPAS-TCR and TCR3d. This mapping is done by the function `get_graph`. If none of the clones in the graph are matched to a CDR3, then the user will have no options to select/highlight.
`show_singletons`	logical, if `show_singletons`=TRUE we plot all vertices. If `show_singletons`=FALSE, we plot only vertices connected by edges.
`node_opacity`	probability, controls the opacity of node colors. Lower values corresponding to more transparent colors.

Value

The output is an `igraph` or `visNetwork` plot.

The size of the vertices increases linearly as the logarithm of the degree of the clonal expansion (number of cells per clone) in the corresponding clones.

Examples

```
# load package input data
data("CDR3ab", package = "ClustIRR")
s <- data.frame(CDR3b = CDR3ab[1:100, "CDR3b"], sample = "A", clone_size = 1)

# run ClustIRR analysis
out <- cluster_irr(s = s)

# get graph
g <- get_graph(clust_irr = out)
```

```
# plot graph with vertices as clones
plot_graph(g, as_visnet=FALSE, show_singletons=TRUE, node_opacity = 0.8)
```

---

tcr3d *CDR3 sequences and their matching epitopes obtained from TCR3d*

Description

data.frame with paired `CDR3a` and `CDR3b` CDR3 sequences and their matching epitopes obtained from TCR3d. The remaining CDR3 columns are set to NA. The antigenic epitopes come from cancer antigens and from viral antigens. For data processing details see the script inst/script/get_tcr3d.R

Usage

```
data(tcr3d)
```

Format

`data.frame` with columns:

1. `CDR3a`: CDR3a amino acid sequence
2. `CDR3b`: CDR3b amino acid sequence
3. `CDR3g`: CDR3g amino acid sequence -> NA
4. `CDR3d`: CDR3d amino acid sequence -> NA
5. `CDR3h`: CDR3h amino acid sequence -> NA
6. `CDR3l`: CDR3l amino acid sequence -> NA
7. `CDR3_species`: CDR3 species (e.g. human, mouse, ...)
8. `Antigen_species`: antigen species
9. `Antigen_gene`: antigen gene
10. `Reference`: Reference ID

Value

data(tcr3d) loads the object tcr3d

Source

TCR3d, June 2024

Examples

```
data("tcr3d")
```

vdjdb *CDR3 sequences and their matching epitopes obtained from VDJdb*

Description

data.frame with unpaired `CDR3a` or `CDR3b` sequences and their matching epitopes obtained from VDJdb. The remaining CDR3 columns are set to NA. For data processing details see the script inst/script/get_vdjdb.R

Usage

```
data(vdjdb)
```

Format

`data.frame` with columns:

1. `CDR3a`: CDR3a amino acid sequence
2. `CDR3b`: CDR3b amino acid sequence
3. `CDR3g`: CDR3g amino acid sequence -> NA
4. `CDR3d`: CDR3d amino acid sequence -> NA
5. `CDR3h`: CDR3h amino acid sequence -> NA
6. `CDR3l`: CDR3l amino acid sequence -> NA
7. `CDR3_species`: CDR3 species (e.g. human, mouse, ...)
8. `Antigen_species`: antigen species
9. `Antigen_gene`: antigen gene
10. `Reference`: Reference (Pubmed ID)

Value

data(vdjdb) loads the object vdjdb

Source

VDJdb, June 2024

Examples

```
data("vdjdb")
```

Index