Package: ClustIRR (via r-universe)

July 5, 2024

Type Package

Title Clustering of immune receptor repertoires

Version 1.3.16

Description ClustIRR is a quantitative method for clustering of immune receptor repertoires (IRRs). The algorithm identifies groups of T or B cell receptors (TCRs or BCRs) with possibly similar specificity directly from the sequences of their complementarity determining regions. ClustIRR uses graphs to visualize the specificity structures of IRRs.

License GPL-3 + file LICENSE

LazyData false

Depends R (>= 4.3.0)

Imports stringdist, BiocParallel, methods, stats, utils, igraph, visNetwork, blaster, grDevices, Biostrings

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

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

Repository https://bioc.r-universe.dev

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

RemoteRef HEAD

RemoteSha 2d05af467c4555c6da9545468a51cdb8bd595764

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*

Description

This algorithm finds groups of TCRs or BCRs with similar specificity. Two clustering strategies are employed:

1. Local clustering
2. Global clustering

Local clustering The aim of local clustering is to find motifs (contiguous k-mers of the CDR sequence) that are overrepresented in repertoire `s` compared to repertoire `r`. This is an outline of the local clustering procedure:

1. Trim CDR3 flanks based on `control$trim_flank_aa`
2. For each motif found in `s` compute the following:
   - motif frequencies in data set `s` (f_s) and `r` (f_r)
   - total number of motifs in data set `s` (n_s) and `r` (n_r)
   - ratio of observed vs. expected motif counts using the following formula: OvE=$(f_s/n_s)/(f_r/n_r)$
   - probability p_i of finding the observed or a larger OvE for motif i given that the null hypothesis is true is computed with the Fisher's exact test
   - if a motif passes the criteria defined in control list, set flag pass$_i$ = T, else pass$_i$ = F

Global clustering The aim of global clustering is to find similar CDR3 sequences in repertoire `s`. This is an outline of the global clustering approaches implemented in ClustIRR:

If `global_smart`=FALSE

For each pair of equal-length CDR3 sequences i and j we compute the Hamming distance d_{ij}. If $d_{ij} \leq$ `global_max_hdist` (user-defined input), then i and j are globally similar.

If `global_smart`=TRUE

We find pairs of CDR3 sequences that satisfy a minimum sequence identity defined by `global_min_identity`. Each pair of sequences are aligned. Then, we compute the length of the longest of the two CDR3 sequences (column `max_len`). Then, we compute four alignment scores: a) `weight` - BLOSUM62 score of the aligned CDR3 sequences; b) `nweight` - BLOSUM62 score of the aligned CDR3 sequences normalized by `max_len`; c) `cweight` - BLOSUM62 score of the aligned CDR3 cores (untrimmed part of the CDR3 sequences); d) `ncweight` - BLOSUM62 score of the aligned CDR3 cores (untrimmed part of the CDR3 sequences) normalized by the length of the longest core sequence.

Alternatively, the user can provide a matrix of globally similar CDR3 sequence pairs, computed by a complementary approachs such as TCRdist.

Usage

```
cluster_irr(
    s,
    r,
    ks = 4,
    cores = 1,
    control = list(global_smart = FALSE,
                   global_max_hdist = 1,
                   global_min_identity = 0.7,
                   local_max_fdr = 0.05,
                   local_min_o = 1,
                   trim_flank_aa = 3,
                   global_pairs = NULL,
                   low_mem = FALSE))
```

Arguments

`s`	data.frame, complementarity determining region 3 (CDR3) amino acid sequences observed in an immune receptor repertoire (IRR). The data.frame can have either one column or two columns: • One column: `s` contains CDR3s from a single chain: *CDR3b*, *CDR3a*, *CDR3g*, *CDR3d*, *CDR3h* or *CDR3l* • Two columns: `s` contains CDR3s from both chains (paired), for instance: – *CDR3b* and *CDR3a* [for $\alpha\beta$ TCRs] – *CDR3g* and *CDR3d* [for $\gamma\delta$ TCRs] – *CDR3h* and *CDR3l* [for heavy/light chain BCRs]
`r`	data.frame, reference (or control) repertoire of CDR3 sequences. Must have the same structure (number of columns and column names) as `s`. If this is not specified or set to NULL, then ClustIRR performs only global clustering using sample `s`
`ks`	integer or integer vector, motif lengths. `ks` = 4 (default)
`cores`	integer, number of CPU cores, `cores` = 1 (default).
`control`	list, a named list of auxiliary parameters to control algorithm's behavior. See the details below: • `global_smart` - logical, should we use smart global clustering based of BLOSUM62 scores (slower but more accurate; default) or less smart global clustering based on Hamming distances (faster but less accurate) • `global_min_identity` - probability, what is the minimum sequence identity between a pair of CDR3 sequences for them to even be considered for global similarity inspection (default = 0.7; 70 percent identity) This input is only used if `global_smart` = TRUE. • `global_max_hdist` - integer, if `global_smart`=FALSE, then `global_max_hdist` defines a Hamming distance (HD) threshold, i.e. two CDR3s as globally similar if their Hamming distance is smaller or equal to `global_max_hdist` HD(a, b) $\leq$ `global_max_hdist`. `global_max_hdist` = 1 (default)

- `local_max_fdr` - numeric, maximum False Discovery Rate (FDR) for the detection of enriched motifs. `local_max_fdr` = 0.05 (default)
- `local_min_o` - numeric, minimum absolute frequency of a motif in the s in order for the motif to be used in the enrichment analysis. `local_min_o` = 1 (default)
- `trim_flank_aa` - integer, how many amino acids should be trimmed from the flanks of all CDR3 sequences (only used for local clustering. `trim_flank_aa` = 3 (default))
- `low_mem` - logical, allows low memory mode for global clustering. This will lead to increase in the CPU time but lead to a lower memory footprint. `low_mem` = `FALSE` (default)
- `global_pairs` - data.frame, pre-computed global pairs. If `global_pairs` is provided by the user, then global clustering is not performed. Instead the CDR3 pairs from `global_pairs` are used as global clustering pairs. `global_pairs` is a data.frame matrix with 4 columns. The first two columns, named, from_cdr3 and to_cdr3 contain pairs of CDR3 sequences that are considered globally similar. The third column, called weight, contains a similarity weight. If weights are not available they should be set to 1. The fourth column, called chain, contains the chain immune receptor in which the CDR3s are found: CDR3b or CDR3a [for $\alpha\beta$ TCRs]; CDR3g or CDR3d [for $\gamma\delta$ TCRs]; or CDR3h or CDR3l [for heavy/ligh chain BCRs].

Value

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

`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.). In the following we who the typical structure of these lists:

- `local` - list, local clustering results
  - `m` - data.frame, motif enrichment results with columns:
    - `motif` - motif sequence
    - `f_s` - observed motif counts in `s`
    - `f_r` - observed motif counts in `r`
    - `n_s` - number of all observed motifs in `s`
    - `n_r` - number of all observed motifs in `r`
    - `k` - motif length
    - `ove` - mean observed/expected relative motif frequency
    - `ove_ci_l95` - 95% confidence intervals of ove (lower boundary)
    - `ove_ci_h95` - 95% confidence intervals of ove (upper boundary)
    - `p_value` - p-value from Fisher's exact test
    - `fdr` - false discovery rate, i.e. adjusted p-value by Benjamini & Hochberg correction
    - `pass` - logical value indicating whether a motifs are enriched (`pass=TRUE`) given the user-defined thresholds in control

  - lp - data.frame, enriched motifs are linked to their original CDR3 sequences and shown as rows in the data.frame with the following columns:
    * cdr3 - CDR3 amino acid sequence
    * cdr3_core - core portion of the CDR3 sequence, obtained by trimming trim_flank_aa amino acids (user- defined parameter). If trim_flank_aa = 0, then cdr3 = cdr3_core
    * motif - enriched motif from cdr3_core
- global - data.frame, global clustering results. Pairs of globally similar CDR3s are shown in each row (analogous to lp). If global_smart=FALSE in the control, then global clustering is done based on Hamming distances and the remaining columns of this data.frame are not important. Else, if global_smart=FALSE, then the remainig columns are relevant, i.e. global similarity scores are shown for the complete CDR3 sequence pairs (column weight) or their core (trimmed) CDR3 sequence part (column cweight). The column max_len stores the the maximum length in each pair of CDR3 sequences, and is 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")
s <- data.frame(CDR3b = CDR3ab[1:100, "CDR3b"], clone_size = 1)
r <- data.frame(CDR3b = CDR3ab[1:500, "CDR3b"], clone_size = 1)

# artificially enrich motif 'RQWW' inside sample dataset
substr(x = s$CDR3b[1:20], start = 6, stop = 9) <- "RQWW"

# add an artificial clonal expansion of two sequences to the sample dataset
s <- rbind(s, data.frame(CDR3b = c("CATSRAAKPDGLRALETQYF",
                                   "CATSRAAKPDRQWWLSTQYF"),
                         clone_size = 10))

# run analysis
out <- cluster_irr(s = s,
                   r = r,
                   ks = 4,
                   cores = 1,
                   control = list(
                       global_smart = TRUE,
                       global_max_hdist = 1,
                       local_max_fdr = 0.05,
                       local_min_o = 1,
                       trim_flank_aa = 3,
                       global_pairs = NULL,
                       low_mem = FALSE))

# output class
```

```
class(out)

# output structure
str(out)

# inspect motif enrichment results
knitr::kable(head(slot(out, "clust")$CDR3b$local$m))

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

# get graph
g <- get_graph(out)

# plot graph
plot_graph(g)

# plot graph as visgraph
plot_graph(g, as_visnet = TRUE)
```

`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.).

Below we provide a detailed description of the slots of `clust_irr`. `clust_irr` objects contain two sublists:

- 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.). In the following we who the typical structure of these lists:
  - `local` - list, local clustering results
    - `m` - data.frame, motif enrichment results with columns:
      - `motif` - motif sequence
      - `f_s` - observed motif counts in `s`
      - `f_r` - observed motif counts in `r`
      - `n_s` - number of all observed motifs in `s`
      - `n_r` - number of all observed motifs in `r`
      - `k` - motif length
      - `ove` - mean observed/expected relative motif frequency
      - `ove_ci_l95` - 95% confidence intervals of ove (lower boundary)
      - `ove_ci_h95` - 95% confidence intervals of ove (upper boundary)
      - `p_value` - p-value from Fisher's exact test

        - fdr - false discovery rate, i.e. adjusted p-value by Benjamini & Hochberg correction
        - pass - logical value indicating whether a motifs are enriched (pass=TRUE) given the user-defined thresholds in control
      * lp - data.frame, enriched motifs are linked to their original CDR3 sequences and shown as rows in the data.frame with the following columns:
        - cdr3 - CDR3 amino acid sequence
        - cdr3_core - core portion of the CDR3 sequence, obtained by trimming trim_flank_aa amino acids (user- defined parameter). If trim_flank_aa = 0, then cdr3 = cdr3_core
        - motif - enriched motif from cdr3_core
    - global - data.frame, global clustering results. Pairs of globally similar CDR3s are shown in each row (analogous to lp). If global_smart=FALSE in the control, then global clustering is done based on Hamming distances and the remaining columns of this data.frame are not important. Else, if global_smart=FALSE, then the remainig columns are relevant, i.e. global similarity scores are shown for the complete CDR3 sequence pairs (column weight) or their core (trimmed) CDR3 sequence part (column cweight). The column max_len stores the the maximum length in each pair of CDR3 sequences, and is 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

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

```
# inputs
data("CDR3ab")
s <- data.frame(CDR3b = CDR3ab[1:1000, "CDR3b"])
r <- data.frame(CDR3b = CDR3ab[1:5000, "CDR3b"])

# controls: auxiliary inputs
control <- list(global_smart = TRUE,
```

```
                global_max_hdist = 1,
                global_min_identity = 0.7,
                local_max_fdr = 0.05,
                local_min_o = 1,
                trim_flank_aa = 3,
                global_pairs = NULL,
                low_mem = FALSE)

# clust_irr S4 object generated by function cluster_irr
clust_irr_output <- cluster_irr(s = s, r = r,ks = 4,cores = 1,control = control)

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

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

`get_graph` *Get graph structure from* `clust_irr` *object*

Description

As input we take a `clust_irr` object generated by the function `cluster_irr`.

From the `clust_irr` object we generate a graph in which the vertices represent clones, and undirected edges are drawn between a pair of vertices if the corresponding clones are locally and/or globally similar (see clustering in the documentation of `cluster_irr`.

The main output is an `igraph` object.

Furthermore, we compare CDR3s from the graph against CDR3 sequences with known epitopes from these databases: VDJdb, McPAS-TCR and TCR3d. The comparison is done based on the edit distance, where the parameter `edit_dist` controls the edit distance threshold (default=0). If the edit distance between two CDR3s is smaller than `edit_dist`, then the database information related to this CDR3 is transfered as vertex attribute to the corresponding vertex.

With the parameter `custom_db`, the user can provide additional databases with CDR3 sequences and their cognate antigens.

Usage

```
get_graph(clust_irr,
                  sample_id,
                  custom_db = NULL,
                  edit_dist = 0)
```

Arguments

clust_irr	S4 object generated by the function cluster_irr
sample_id	character, name of the repertoire. It will be appended to each node's name. If missing, a random sample_id will be generated (e.g. S10)
custom_db	data.frame, custom database mapping CDR3 sequences to their cognate antigens. The structure of custom_db must be identical to e.g. data(vdjdb). This is an optional input that allows us to annotate the CDR3s in our data. If custom_db is not provided, ClustIRR will use the internal databases. Else, ClustIRR will use both the internal and user- provided databases.
edit_dist	integer, edit distance threshold. CDR3 sequences from the graph will be matched with CDR3 sequences with known antigenic specificities (from a database). If their edit distance is lower or equal to edit_dist, then we have a match. Default edit_dist = 0, i.e. only identical CDR3s are matched.

Value

The main output is a list. The list contains an igraph object. The graph vertices and edges contain attributes. Furthermore, it contains a data.frame in which rows are clones (vertices) in the graph. Finally, 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")
s <- data.frame(CDR3b = CDR3ab[1:100, "CDR3b"], clone_size = 1)
r <- data.frame(CDR3b = CDR3ab[1:5000, "CDR3b"], clone_size = 1)

# artificially enrich motif 'RWGW' inside sample dataset
substr(x = s$CDR3b[1:20], start = 6, stop = 9) <- "RWGW"

# add an artificial clonal expansion of two sequences to the sample dataset
s <- rbind(s, data.frame(CDR3b = c("CATSRADKPDGLDALETQYF",
                                   "CATSRAAKPDGLAALSTQYF"),
                         clone_size = 5))

# run ClustIRR analysis
out <- cluster_irr(s = s,
                   r = r,
                   ks = 4,
                   cores = 1,
                   control = list(trim_flank_aa = 3))

# get graph
g <- get_graph(out)

names(g)
```

get_joint_graph *Joins two graphs obtained from two or more* clust_irr *objects*

Description

As input we take at least two clust_irr objects generated by the function cluster_irr.

From each clust_irr object we generate a graph (with the function get_graph) in which the vertices represent clones, and undirected edges are drawn between a pair of vertices if the corresponding clones are locally and/or globally similar (see definitions of local/global clustering in the documentation of cluster_irr.

Additionally, get_joint_graph performs the following operation between each pair of graphs:

First, it merges the vertices. If graphs a and b have |a| and |b| vertices, then the joint graph has |a|+|b| vertices, regardless of whether exactly the same clone (vertex) is found in both graphs.

Second, it performs global clustering between graph pairs. If two clones (graph vertices) have similar CDR3 sequences, then the vertices are connected by an edge. The same global clustering strategy (controls) are applied as the ones used to build the individual clust_irr objects. If different clust_irr objects are generated with different controls, then the graph merging algorithm will exit with an error.

The main output is an igraph object.

Third, we compare CDR3s from the graph against CDR3 sequences with known epitopes from these databases: VDJdb, McPAS-TCR and TCR3d. The comparison is done based on the edit distance, where the parameter edit_dist controls the edit distance threshold (default=0). If the edit distance between two CDR3s is smaller than edit_dist, then the database information related to this CDR3 is transfered as vertex attribute to the corresponding vertex.

With the parameter custom_db, the user can provide additional databases with CDR3 sequences and their cognate antigens.

Usage

```
get_joint_graph(clust_irrs,
                cores = 1,
                custom_db = NULL,
                edit_dist = 0)
```

Arguments

clust_irrs	A list of at least two S4 objects generated with the function cluster_irr
cores	integer, number of computer cores to use (default = 1)
custom_db	data.frame, custom database with CDR3 sequences and their matching antigens. The structure of custom_db must be identical to data(vdjdb). This is an optional input to annotate the clones. If custom_db is not provided, ClustIRR will use internal databases. See description below.

edit_dist — integer, edit distance threshold. CDR3 sequences from the graph will be matched with CDR3 sequences with known antigenic specificities (from a database). If their edit distance is lower or equal to edit_dist, then we have a match. Default edit_dist = 0, i.e. only identical CDR3s are matched.

Value

The main output of this function is an igraph object. This object represents a joint graph of the individual graphs contained as elements in the input clust_irrs. The graph nodes and edges contain many attributes which are described in the description section.

One additional output is a data.frame in which rows are clones (vertices) in the joint graph.

Examples

```
# load package input data
data("CDR3ab")
s_1 <- data.frame(CDR3b = CDR3ab[1:100, "CDR3b"])
s_2 <- data.frame(CDR3b = CDR3ab[101:200, "CDR3b"])
r <- data.frame(CDR3b = CDR3ab[1:500, "CDR3b"])


# run 1st analysis -> clust_irr object
o_1 <- cluster_irr(s = s_1, r = r, ks = 4)

# run 2nd analysis -> clust_irr object
o_2 <- cluster_irr(s = s_2, r = r, ks = 4)

# join clust_irr objects in a list
clust_irrs <- c(o_1, o_2)
names(clust_irrs) <- c("C1", "C2")

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

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")
s <- data.frame(CDR3b = CDR3ab[1:100, "CDR3b"], clone_size = 1)
r <- data.frame(CDR3b = CDR3ab[1:500, "CDR3b"], clone_size = 1)

# artificially enrich motif 'RWGW' inside sample dataset
substr(x = s$CDR3b[1:20], start = 6, stop = 9) <- "RWGW"

# add an artificial clonal expansion of two sequences to the sample dataset
s <- rbind(s, data.frame(CDR3b = c("CATSRADKPDGLDALETQYF",
                                   "CATSRAAKPDGLAALSTQYF"),
                         clone_size = 5))

# run analysis
ci <- cluster_irr(s = s,
                  r = r,
                  ks = 4,
                  cores = 1,
                  control = list(trim_flank_aa = 3))

g <- get_graph(clust_irr = ci)

# 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