Package: ClustIRR (via r-universe)

May 31, 2026

Type Package

Title Clustering of Immune Receptor Repertoires

Version 1.10.0

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

Depends R (>= 4.3.0)

Imports grDevices, igraph, methods, Rcpp (>= 0.12.0), RcppParallel (>= 5.0.1), reshape2, rstan (>= 2.18.1), rstantools (>= 2.4.0), stats, stringdist, utils, posterior, visNetwork, dplyr, tidyr, ggplot2, ggforce, scales, msa, Biostrings, RADanalysis, ggseqlogo, rBLAST

Suggests BiocStyle, knitr, testthat, ggrepel, patchwork, htmlwidgets

SystemRequirements GNU make, ncbi-blast+

Encoding UTF-8

LazyData false

NeedsCompilation yes

biocViews Clustering, ImmunoOncology, SingleCell, Software, Classification, Bayesian, BiomedicalInformatics, ImmunoOncology, MathematicalBiology

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)

Config/pak/sysreqs cmake libfontconfig1-dev libfreetype6-dev libglpk-dev make libicu-dev libuv1-dev libxml2-dev libssl-dev zlib1g-dev

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

Date/Publication 2026-04-28 13:01:37 UTC

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

RemoteRef RELEASE_3_23

RemoteSha 0634e77846702f14a6f723e03fa2c9fa662c6260

Contents

BLOSUM62 *BLOSUM62 matrix*

Description

Predefined scoring matrix for amino acid or nucleoitide alignments.

Usage

```
data("BLOSUM62")
```

Format

BLOSUM62 is a square symmetric matrix. Rows and columns are identical single letters, representing nucleotide or amino acid. Elements are integer coefficients (substitution scores).

Details

BLOSUM62 was obtained from NCBI (the same matrix used by the stand- alone BLAST software).

Source

See https://ftp.ncbi.nih.gov/blast/matrices/BLOSUM62

References

See https://ftp.ncbi.nih.gov/blast/matrices/BLOSUM62

Examples

```
data(BLOSUM62, package = "ClustIRR")
BLOSUM62
```

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

```
# only run if blast is installed
if(rBLAST::has_blast()) {

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

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

    # output class
    class(c)

    # output structure
    str(c)

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

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

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

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

Description

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

Usage

```
clustirr(s,
         meta = NULL,
         control = list(blast_gmi = 0.8,
                        blast_cores = 1,
                        trim_flank_aa = 3,
                        db_dist = 0,
                        db_custom = NULL,
                        knn = FALSE,
                        k = 30))
```

Arguments

`s` a data.frame consisting of one or more IRRs. Each row is a clone of a given IRR with the following columns (clone features):

- `sample`: names of the repertoires (e.g. 'A', 'B', etc.)
- `clone_size`: cell count in the clone (=clonal expansion)
- `CDR3?`: amino acid CDR3 sequence. Replace '?' with the appropriate name of the immmune receptor 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]

`meta` data.frame with meta-data for each clone, which may contain clone-specific data, such as, V/J genes, cell-type (e.g. CD8+, CD4+), nut also repertoire-specific data, such as, biological condition, HLA type, age, etc. This data will be used to annotate the graph nodes and help downstream analyses.

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

- `blast_gmi`: the minimum sequence identity between a pair of CDR3 sequences for them to even be considered for alignment and scoring (default = 0.8; 80 percent identity).
- `blast_cores`: the number of cores to use with BLAST (default = 1).
- `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 `db_dist` (default = 0), then we annotate the CDR3 from `s` with the specificity of the database CDR3 sequence.
- `knn` and `k`: Should ClustIRR use a k-nearest-neighbor (KNN) approach to construct the graphs? The default is `knn = FALSE`. If `knn = TRUE`, `k` specifies the maximum number of neighbors each node can have per chain. For example, with paired-chain data and `k = 1`, a node can have at most two neighbors (one per chain). If the analysis involves graph joining, then for each pair of graphs a node may have up to `k` neighbors per chain.

Details

ClustIRR performs the following steps.

1. Compute similarities between clones within each repertoire
2. Construct a graph from each TCR repertoire
3. Construct a joint similarity graph (J)
4. Detect communities in J
5. Analyze Differential Community Occupancy (DCO)
   - Between individual TCR repertoires with model M
   - Between groups of TCR repertoires from biological conditions with model M_h
6. Inspect results

the function `clustirr` performs the steps 1. to 3.

Value

The output is a list with the following elements.

- graph: the resulting `igraph` object
- clust_irrs: list of clust_irr objects for each repertoire (sample)
  Each element is an `S4` object of class `clust_irr`. This object contains two sublists:

  - clust, list, contains clustering results for each receptor 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)

- multigraph: logical variable multigraph, which is set to TRUE if the graph is a joint graph made up of two or more repertoires (samples) and FALSE if the graph contains only one repertoire

Examples

```
# only run if blast is installed
if(rBLAST::has_blast()) {
    # load package input data
    data("CDR3ab", package = "ClustIRR")
    s <- data.frame(CDR3b = CDR3ab[1:100, "CDR3b"], sample = "A", clone_size=1)

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

    # output class
    class(c)

    # output structure
    str(c)
}
```

Datasets *Datasets* CDR3ab, D1 *and* D2 *with TCR$\alpha\beta$ mock repertoires*

Description

TCR$\alpha\beta$ repertoire with 10,000 T-cells (rows). Each T-cell has the following features: amino acid sequences of their complementarity determining region 3 (CDR3); and variable (V) and joining (J) gene names for TCR chains α and β.

Important remark: this is a mock dataset, all CDR3 sequences and the genes were sampled from a larger set of CDR3β sequences and genes of naive CD8+ T cells in humans.

About dataset D1:

We used this data to create dataset D1: three TCR$\alpha\beta$ repertoires a, b, and c, each with 500 TCR clones. We simulated clonal expansion with increasing degree in TCR repertoires b and c. The TCR repertoires are stored as element of a list. For each TCR repertoires we have a metadata: ma, mb, and mc.

About dataset D2:

We used b and c from D1 to create dataset D2. Three TCR$\alpha\beta$ repertoires (replicates) were generated from two biological conditions: A (A1, A2, A3) and C (C1, C2, C3), each with 2,000 T-cells.

Usage

```
# For the raw data with 10,000 TCR clones
data(CDR3ab)

# For dataset D1
data(D1)

# For dataset D2
data(D2)
```

Format

CDR3ab is a `data.frame` with rows as TCR clones and 6 columns

- `CDR3a`: CDR3α amino acid sequence
- `TRAV`: variable (V) gene of TCRα
- `TRAV`: joining (J) gene of TCRα
- `CDR3b`: CDR3β amino acid sequence
- `TRBV`: variable (V) gene of TCRβ
- `TRBV`: joining (J) gene of TCRβ

Value

data(CDR3ab) loads the object CDR3ab, which is a data.frame with six columns (3 for TCRα and 3 for TCRβ) and rows for each TCR clone (see details).

Source

GLIPH version 2

Examples

```
data("CDR3ab")
data("D1")
data("D2")
```

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, compute_delta=TRUE, groups = NA)
```

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_cores = 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.

`compute_delta`
: should delta be computed by the Stan model? This may be take up extra memory.

`groups`
: vector with integers ≥ 1, one for each repertoire (column in community_occupancy_matrix). This specifies the biological group of each repertoire (e.g. for cancer repertoire we may specify the index 1, and for normal repertoires the index 2). If this vector is specified, ClustIRR will employ a hierarchical model, modeling the dependence between the repertoires within each group. Else (which is the default setting in ClustIRR), ClustIRR will treat the repertoires as independent samples by employing a simpler model.

Value

The output is a list with the folling elements:

`fit`
: model fit (stan object)

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
: mcmc configuration inputs provided as list.

compute_delta
: the input compute_delta.

groups
: the input groups.

Examples

```
# only run if blast is installed
if(rBLAST::has_blast()) {
    # 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 <- clustirr(s = rbind(a, b))

    # detect communities
    gcd <- detect_communities(graph = c$graph,
                              algorithm = "leiden",
                              resolution = 1,
                              iterations = 100,
                              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$community_summary$wide)

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

    # differential community occupancy analysis
```

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

    names(dco)
}
```

`decode_all_communities` *Decode all graph communities*

Description

Given a graph based on which we have detected communities (with the function `detect_community`), this function applies `decode_communities` to **all** communities in the graph.

Each community is partitioned according to user-defined filters on edges (`edge_filter`) and nodes (`node_filter`). This allows extraction of subgraphs, cliques, and connected components across the full set of communities in the graph.

Usage

```
decode_all_communities(graph, edge_filter, node_filter)
```

Arguments

`graph`	`igraph` object that has been analyzed by graph-based community detection methods as implemented in `detect_community`. Must have a vertex attribute named `community`.
`edge_filter`	data.frame with edge filters. The data.frame has three columns:

- name: edge attribute name
- value: edge attribute value (threshold)
- operation: logical operation that tells ClustIRR which edge attribute values should pass the filter. Possible operations: "<", ">", ">=", "<=", "==" and "!=".

`node_filter`	data.frame with node filters. Groups of nodes that have the same attribute values among **ALL** provided attributes will be treated as a subcomponent.

Details

Internally, this function iterates over all unique community IDs found in the vertex attribute `community`, and applies `decode_communities`.

- The `edge_filter` controls which edges are retained based on their attributes.
- The `node_filter` groups nodes with shared attributes into subcomponents.

Value

A named list of results, one per community. Each element of the list is the output of `decode_communities`, containing:

- community_graph: "filtered" igraph object for the community
- component_stats: data.frame with summary about each connected component
- node_summary: data.frame with summary about each node

Examples

```
# only run if blast is installed
if(rBLAST::has_blast()) {
    # 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 <- clustirr(s = rbind(a, b))

    # detect communities
    gcd <- detect_communities(graph = c$graph,
                              algorithm = "leiden",
                              resolution = 1,
                              iterations = 100,
                              chains = c("CDR3a", "CDR3b"))

    # Construct edge and node filters
    edge_filter <- rbind(data.frame(name = "nweight", value = 8,
                                    operation = ">="),
                         data.frame(name = "ncweight", value = 8,
                                    operation = ">="))
    node_filter <- data.frame(name = "Ag_gene")

    # Decode all communities at once
    all_decoded <- decode_all_communities(graph = gcd$graph,
                                          edge_filter = edge_filter,
                                          node_filter = node_filter)

    # Inspect one decoded community
    names(all_decoded)
    str(all_decoded[[1]]$component_stats)
}
```

`decode_community` *Decode graph communities*

Description

Given a graph based on which we have detected communities (with the function `detect_communities`), and a community ID, the function will try to partition the community nodes according to user-defined filters: edge and node filters.

Usage

```
decode_community(community_id, graph, edge_filter, node_filter)
```

Arguments

`graph`	`igraph` object that has been analyzed by graph-based community detection methods as implemented in `detect_communities`
`community_id`	which community should be decoded?
`edge_filter`	data.frame with edge filters. The deta.frame has three columns:

- name: edge attribute name
- value: edge attribute value (threshold)
- operation: logical operation that tells ClustIRR which edge attribute values should pass the filter. Possible operations: "<", ">", ">=", "<=", "==" and "!=".

`node_filter`	a vector with node attributes. Groups of nodes that have the same attribute values among **ALL** provided attributes will be treated as a subcomponent.

Details

How to decode a community?

For instance, the user may only be interested in retaining edges with core edge weight > 4; or making sure that nodes that have same 'cell_type' (node meta datafrom) are grouped together. Or the user might want to treat all nodes that have the same V, D and J gene names and HLA types as subgroups, in which case all edges between nodes that do not share the same sets of attributes are dicarded.

Based on these `filters`, ClustIRR will reformat the edges in the selected community and then find **connected components** in the resulting graph.

Value

The output is a list with the following elements

- community_graph: "filtered" igraph object
- component_stats: data.frame with summary about each connected component
- node_summary: data.frame with summary about each node

Examples

```
# only run if blast is installed
if(rBLAST::has_blast()) {
    # 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 <- clustirr(s = rbind(a, b))

    # detect communities
    gcd <- detect_communities(graph = c$graph,
                              algorithm = "leiden",
                              resolution = 1,
                              iterations = 100,
                              chains = c("CDR3a", "CDR3b"))

    # We "decompose" the communities in the gcd object using decode_community
    # using attributes of the edges (edge_filter) and nodes (node_filter).
    # We can pick from these edge attributes and create filters:
    library(igraph)
    edge_attr_names(graph = gcd$graph)

    # For instance, the following edge-filter will instruct ClustIRR to keep
    # edges with: edge attributes: nweight>=8 \bold{AND} ncweight>=8
    edge_filter <- rbind(data.frame(name = "nweight", value = 8,
                                    operation = ">="),
                         data.frame(name = "ncweight", value = 8,
                                    operation = ">="))

    # In addition, we can construct filters using the following node attributes:
    vertex_attr_names(graph = gcd$graph)

    # The following node-filter will instruct ClustIRR to retain edges
    # between nodes that have shared node attributed with respect to ALL
    # of the following node attributes:
    node_filter <- data.frame(name = "Ag_gene")

    # Lets inspect community with ID = 1.
    c1 <- decode_community(community_id = 1,
                           graph = gcd$graph,
                           edge_filter = edge_filter,
                           node_filter = node_filter)
```

```
    # Plot resulting igraph
    par(mar = c(0, 0, 0, 0))
    plot(c1$community_graph, vertex.size = 10)

    # Now look at node attributes
    as_data_frame(x = c1$community_graph, what = "vertices")[,c("name",
                                                                "component_id",
                                                                "CDR3b",
                                                                "CDR3a",
                                                                "Ag_gene")]

    str(c1$component_stats)

    str(c1$node_summary)
}
```

`detect_communities` *Graph-based community detection (GCD)*

Description

Graph-based community detection in graphs constructed by `get_graph` or `get_joint_graph`.

Usage

```
detect_communities(graph,
                   algorithm = "leiden",
                   metric = "average",
                   resolution = 1,
                   iterations = 100,
                   chains)
```

Arguments

`graph`	`igraph` object
`algorithm`	graph-based community detection (GCD) method: leiden (default), louvain or infomap.
`metric`	possible metrics: "average" (default) or "max".
`resolution`	clustering resolution (default = 1) for GCD.
`iterations`	clustering iterations (default = 100) for GCD.
`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, Leiden or InfoMap, to identify `communities` of nodes that have high density of edges among each other, and low density of edges with nodes outside the community.

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.

`graph_structure_quality`
: graph modularity and quality (only for Leiden) measure of the strength of division of the graph into communities.

`input_config`
: list, inputs provided as list.

Examples

```
# only run if blast is installed
if(rBLAST::has_blast()) {
    # 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 <- clustirr(s = rbind(a, b))

    # detect communities
    gcd <- detect_communities(graph = c$graph,
                              algorithm = "leiden",
                              metric = "average",
                              resolution = 1,
                              iterations = 100,
                              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_ag_gene_hits` *Annotate antigen gene hits in node summary*

Description

Annotates clones (nodes) in the node summary output from the function `detect_communities` with specificity to given antigen gene name. Adds new binary columns indicating hits and computes antigen-specific cellular statistics per community and sample.

Usage

```
get_ag_gene_hits(node_summary,
                 db = "vdjdb",
                 ag_gene)
```

Arguments

`node_summary`	Node summary data.frame from `detect_communities`
`db`	Annotation database (e.g., "vdjdb", "mcpas", "tcr3d")
`ag_gene`	Antigen gene name (e.g., "MLANA", "gp100", "Spike")

Details

Searches for antigen gene matches (e.g., "MLANA" or "gp100") in annotation columns from specified databases (e.g., "vdjdb"). For each match, adds a new column to `node_summary` (1 = hit, 0 = no hit) and computes:

- Repertoire-level: Total cells/clones per sample
- Community-level: Cells/clones per community
- Antigen-specific: Cells/clones per antigen-community

Requires `node_summary` from `detect_communities` with database annotation columns).

Value

List containing:

`node_summary`	Input with added antigen hit columns. See `new_columns` for the names of the added columns
`new_columns`	Names of added columns

table_summary Aggregated data.frame with columns:

- sample, community, ag_key
- rep_cells, rep_clones (repertoire totals)
- com_cells, com_clones (community totals)
- ag_cells, ag_clones (antigen-specific counts)

Examples

```
# only run if blast is installed
if(rBLAST::has_blast()) {
    # 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 <- clustirr(s = rbind(a, b))

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

    ag <- get_ag_gene_hits(node_summary = gcd$node_summary,
                           db = "vdjdb",
                           ag_gene = "Spike")

    head(ag)
}
```

---

get_ag_species_hits *Annotate antigen species hits in node summary*

Description

Annotates clones (nodes) in the node summary output from the function detect_communities with specificity to given antigen species. Adds new binary columns indicating hits and computes antigen-specific cellular statistics per community and sample.

Usage

```
get_ag_species_hits(node_summary,
                    db = "vdjdb",
                    ag_species)
```

Arguments

node_summary	Node summary data.frame from detect_communities
db	Annotation database (e.g., "vdjdb", "mcpas", "tcr3d")
ag_species	Antigen species (e.g., "EBV", "CMV", "SARS-CoV-2")

Details

Searches for antigen species matches (e.g., "EBV" or "CMV") in annotation columns from specified databases (e.g., "vdjdb"). For each match, adds a new column to node_summary (1 = hit, 0 = no hit) and computes:

- Repertoire-level: Total cells/clones per sample
- Community-level: Cells/clones per community
- Antigen-specific: Cells/clones per antigen-community

Requires node_summary from detect_communities with database annotation columns).

Value

List containing:

node_summary	Input with added antigen hit columns. See new_columns for the names of the added columns
new_columns	Names of added columns
table_summary	Aggregated data.frame with columns: • sample, community, ag_key • rep_cells, rep_clones (repertoire totals) • com_cells, com_clones (community totals) • ag_cells, ag_clones (antigen-specific counts)

Examples

```
# only run if blast is installed
if(rBLAST::has_blast()) {
    # 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 <- clustirr(s = rbind(a, b))

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

    ag <- get_ag_species_hits(node_summary = gcd$node_summary,
                              db = "vdjdb",
                              ag_species = "EBV")

    head(ag)
}
```

`get_beta_cprob_ag` *Visualize cumulative probability of β means for antigen-specific communities*

Description

Visualize the cumulative probability distribution of β means for each repertoire. The function compares antigen-specific communities against all communities by plotting cumulative probability curves.

Usage

```
get_beta_cprob_ag(beta,
                  node_summary,
                  ag,
                  ag_species = TRUE,
                  db = "vdjdb")
```

Arguments

`beta`	`beta` data.frame
`node_summary`	`node_summary` data.frame
`ag`	antigen species/gene, character, e.g. "EBV", "CMV", or "MLANA"
`ag_species`	logical, is the antigen a species (TRUE) or gene (FALSE)
`db`	annotation database, character, e.g. "vdjdb"

Details

The user has to provide an antigen species (e.g. ag = "EBV" and ag_species=TRUE) or an antigen gene (e.g. ag = "MLANA" and ag_species=FALSE). Furthermore, the user has to provide node_summary (data.frame created by the function detect_communities) and beta data.frame which is part of posterior_summary generated by the function dco.

The function identifies antigen-specific communities using the selected annotation database db, such as "vdjdb", "mcpas", or "tcr3d". Perfect matches between CDR3 sequences in the input and in the annotation database are used for annotation.

Cumulative probability curves are computed separately for:

- antigen-specific communities
- all communities

These are shown as solid and dashed lines, respectively, allowing comparison of their distributions across samples.

Value

A list containing:

- data: data.frame with cumulative probabilities for antigen-specific and all communities
- g: a ggplot object showing cumulative probability curves

Examples

```
# only run if blast is installed
if(rBLAST::has_blast()) {
    # 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 <- clustirr(s = rbind(a, b))

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

    # differential community occupancy analysis
```

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

    # generate cumulative probability plot
    res <- get_beta_cprob_ag(beta = dco$posterior_summary$beta,
                             node_summary = gcd$node_summary,
                             ag = "EBV",
                             ag_species = TRUE,
                             db = "vdjdb")

    # access plot
    res$g
}
```

`get_beta_violin_ag` *Visualize distribution of β means in each repertoire as violin plots*

Description

Visualize the β means as violin plots, representing relative community occupancies for individual repertoires. At the same time, annotate the communities (dots) based on their specificity.

Usage

```
get_beta_violin_ag(beta,
                   node_summary,
                   ag,
                   ag_species,
                   db = "vdjdb")
```

Arguments

`beta`	`beta` data.frame
`node_summary`	`node_summary` data.frame
`ag`	antigen species/gene, character, e.g. "EBV", "CMV", or "MLANA"
`ag_species`	is the antigen a species (TRUE) or gene (FALSE)
`db`	annotation database, character, e.g. "vdjdb"

Details

The user has to provide an antigen species (e.g. `ag` = "EBV" and `ag_species`=TRUE) or an antigen gene (e.g. `ag` = "MLANA" and `ag_species`=FALSE). Furthermore, the user has to provide nodes (`node_summary` data.frame created by the function `detect_communities`) and `beta` data.frame which is part of `posterior_summary` generated by the function `dco`.

The user can also select an annotation database `db`, such as "vdjdb", "mcpas" or "tcr3d". We will look for perfect matches between CDR3 sequences in the input and in the annotation database for annotation.

Value

The output is a violin ggplot.

Examples

```
# only run if blast is installed
if(rBLAST::has_blast()) {
    # 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 <- clustirr(s = rbind(a, b))

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

    # differential community occupancy analysis
    dco <- dco(community_occupancy_matrix = gcd$community_occupancy_matrix)

    # generate beta violin plots
    v <- get_beta_violin_ag(beta = dco$posterior_summary$beta,
                            node_summary = gcd$node_summary,
                            ag = "EBV",
                            ag_species = TRUE,
                            db = "vdjdb")
}
```

---

get_cdr3_motifs *Generate CDR3 motif for communities*

Description

This function extracts CDR3 motifs from the node summary of clonotypes in a specific community, and generates motif logos for each chain using sequence alignment and amino acid frequency calculations. It allows the removal of gaps in the alignment based on a user-defined probability threshold.

Usage

```
get_cdr3_motifs(node_summary, community_id, gap_remove_prob = 0.8)
```

Arguments

node_summary	A data frame containing node information generated by the function detect_communities. The rows represent clonotypes, and the columns include chains (e.g., 'CDR3a', 'CDR3b') and other attributes such as 'clone_size'.
community_id	A numeric or character value representing the community ID for which CDR3 motifs should be extracted.
gap_remove_prob	A numeric value between 0 and 1 specifying the probability threshold for removing gaps in the sequence alignment. The default is 0.8. Positions with a gap frequency above this threshold will have gaps excluded.

Details

The function performs the following steps:

1. Extracts CDR3 sequences for each chain from the node summary.
2. Aligns the CDR3 sequences using Clustal Omega and computes position-specific amino acid frequencies.
3. Generates motif logos for each chain, with the option to remove gaps based on the specified threshold.
4. The motif logos are arranged in a single row using the 'patchwork' package.

Value

A 'ggplot2' object that displays a series of motif logos for each chain in the specified community. The logos show the amino acid frequency at each position in the CDR3 sequence, with optional gap removal based on the 'gap_remove_prob'.

Examples

```
# Create a mock node summary data frame
node_summary <- data.frame(
  community = rep(1, 4),
  CDR3a = c("CASSLAGTDTQYF", "CASSLAGTDTQYF", "CASSLAGTDTQYF", "CASSLAGTDTQYF"),
  CDR3b = c("CASSLAGTDTQYF", "CASSLAGTDTQYF", "CASSLAGTDTQYF", "CASSLAGTDTQYF"),
  clone_size = c(100, 150, 200, 50)
)

# Call the function to generate motif logos for community 1
motifs <- get_cdr3_motifs(node_summary, community_id = 1, gap_remove_prob = 0.8)

# Display the resulting motif logos
plot(motifs)
```

`get_community_feature_purity` *Compute community purity with respect to a node feature*

Description

Compute the purity of graph communities with respect to a node feature.

Given a `node_summary` object returned by `detect_communities` containing a `community` column, this function evaluates how homogeneous each community is with respect to a specified node feature.

If the feature is numeric (e.g., gene expression, clone size), purity is measured using the coefficient of variation (CV). If the feature is categorical (e.g., tissue type, sample, or cell type), purity is measured using Gini impurity (GI) and Shannon's entropy (H). For binary categorical features, the signed dominance score (D) is additionally reported to indicate which class dominates a community.

Usage

```
get_community_feature_purity(node_summary, node_feature)
```

Arguments

`node_summary`	A `data.frame` returned by `detect_communities` containing node attributes and a column named `community`.
`node_feature`	Character scalar specifying the node attribute for which community purity should be evaluated. The feature must be either numeric or categorical.

Details

The function verifies that `node_summary` contains a `community` column and that the requested `node_feature` exists. The feature type is determined automatically.

For categorical features, purity metrics are computed per community:

- **Gini impurity (GI)** measures the probability that two randomly selected nodes from the community have different feature labels.
- **Shannon's entropy (H)** measures how evenly feature labels are distributed within the community.
- **Signed dominance (D)** is computed when the feature has exactly two classes and quantifies which class dominates the community.

Lower values of GI and H indicate higher purity (i.e., communities dominated by a single feature value). The dominance score ranges from -1 to 1, where positive and negative values indicate dominance of one or the other class.

For numeric features, the coefficient of variation (CV) is computed:

- **Mean** and **standard deviation (sd)** are computed per community.
- **CV = sd / mean** measures the relative dispersion of the feature within the community.

Lower CV values indicate more homogeneous communities with respect to the numeric feature.

Value

A `data.frame` summarizing purity statistics for each community.

For categorical features the returned columns are:

- `community` – community identifier
- `GI` – Gini impurity
- `H` – Shannon's entropy
- `D` – signed dominance score (only meaningful for binary features)
- `n` – number of nodes in the community

For numeric features the returned columns are:

- `community` – community identifier
- `mean` – mean feature value
- `sd` – standard deviation
- `n` – number of nodes in the community
- `cv` – coefficient of variation

Examples

```
# only run if blast is installed
if(rBLAST::has_blast()) {
    # load package input data
    data("D1", package = "ClustIRR")

    # run ClustIRR analysis
    c <- clustirr(s = rbind(D1$a, D1$b), meta = rbind(D1$ma, D1$mb))

    # detect communities
    gcd <- detect_communities(graph = c$graph,
                              algorithm = "leiden",
                              metric = "average",
                              resolution = 1,
                              iterations = 100,
                              chains = c("CDR3a", "CDR3b"))

    # categorical feature
    get_community_feature_purity(node_summary = gcd$node_summary,
                                 node_feature = "HLA_A")

    # another categorical feature
    get_community_feature_purity(node_summary = gcd$node_summary,
                                 node_feature = "TRAV")

    # numeric feature
    get_community_feature_purity(node_summary = gcd$node_summary,
                                 node_feature = "age")
}
```

`get_community_feature_stats` *Compute descriptive statistics of a community node feature*

Description

Given a `node_summary` object containing community assignments and node- level attributes, this function computes descriptive statistics of a specified node feature within a single community.

Usage

```
get_community_feature_stats(node_summary, node_feature, community_id)
```

Arguments

`node_summary`	A data.frame containing node-level attributes and a `community` column.
`node_feature`	Character scalar specifying the node attribute for which statistics should be computed.
`community_id`	Numeric community ID

Details

The function requires a `node_summary` data.frame containing a `community` column and a valid `node_feature` column.

The type of statistics returned depends on whether the node feature is numeric or categorical:

- For numeric features, the function computes mean, median, sum, and the number of nodes.
- For categorical (character, factor, or logical) features, the function computes counts and proportions for each feature level.

The community identifier is inferred from the `community` column and is assumed to be constant within `node_summary`.

Value

A `data.frame` summarizing feature statistics for the given community:

- For numeric features: columns `community`, `feature`, `feature_mean`, `feature_median`, `feature_sum`, `feature_type`, and `n`.
- For categorical features: columns `community`, `feature`, `feature_count`, `feature_prop`, `feature_type`, and `n`.

Examples

```
# only run if blast is installed
if(rBLAST::has_blast()) {
    # load package input data
    data("D1", package = "ClustIRR")

    # run ClustIRR analysis
    c <- clustirr(s = rbind(D1$a, D1$b), meta = rbind(D1$ma, D1$mb))

    # detect communities
    gcd <- detect_communities(graph = c$graph,
                              algorithm = "leiden",
                              metric = "average",
                              resolution = 1,
                              iterations = 100,
                              chains = c("CDR3a", "CDR3b"))

    # numeric feature
    get_community_feature_stats(
        community_id = 1,
        node_summary = gcd$node_summary,
        node_feature = "age")

    # categorical feature
    get_community_feature_stats(
        community_id = 1,
        node_summary = gcd$node_summary,
        node_feature = "HLA_A")
}
```

---

get_cosine_similarity *Compute and Visualize Cosine Similarity (CS)*

Description

This function calculates pairwise CSs between columns (repertoires) of the community occupancy matrix and generates a labeled heatmap visualization. Here CS measures the cosine of the angle between vectors of cell counts (positive integers or zero), providing a value between 0 (perfect dissimilarity) and 1 (perfect similarity).

Usage

```
get_cosine_similarity(com)
```

Arguments

com	Community occupancy matrix where columns represent repertoires for CS computation. Rows should are communities. Entries are the number of cells in each repertoire and community.

Details

The function performs three main operations:

1. Computes CS between all column pairs using:

$$\text{CS}(x, y) = \frac{x \cdot y}{||x|| \cdot ||y||}$$

2. Creates a heatmap with CS values annotated in each cell

Red and blue tiles in the heatmap indicates high and low CS, respectively.

Value

A list containing two elements:

- `g`: A `ggplot2` heatmap displaying the CS
- `cs`: A data frame in long format with columns:
  - `i`: First repertoire
  - `j`: Second repertoire
  - `CS`: CS (rounded to 2 decimals in plot labels)

Examples

```
# Create a sample matrix
mat <- matrix(rpois(n=15, lambda = 4), nrow = 5,
              dimnames = list(NULL, c("A", "B", "C")))

# Compute similarities and plot
result <- get_cosine_similarity(mat)

# Display heatmap
print(result$g)

# Inspect similarity values
head(result$cs)
```

---

get_honeycombs *Generate honycomb plot: visualize community occupancy of pairs of immune receptor repertoires*

Description

Use the `community_occupancy_matrix` generated by the function `detect_communities` to generate honeycomb plots for each pair of repertoires. In each plot, we will show communities (rows in the matric `community_occupancy_matrix`) as dots and their intensities in a pair of repertoires (x-axis and y-axis). The density of dots is encoded by the color of the honeycomb-like hexagons.

Usage

```
get_honeycombs(com)
```

Arguments

com	community_occupancy_matrix, matrix generated by detect_communities

Details

Use the community_occupancy_matrix generated by the function detect_communities to generate honeycomb plots for each pair of repertoires. In each plot, we will show communities (rows in the matric community_occupancy_matrix) as dots and their intensities in a pair of repertoires (x-axis and y-axis). The density of dots is encoded by the color of the honeycomb-like hexagons.

Value

The output is a list with ggplots. Given n repertoires (columns in input community_occupancy_matrix), it will generate n*(n-1)/2 plots. You can arrange the ggplots (or a portion of them) in any shape e.g. with the R-package patchwork.

Examples

```
# only run if blast is installed
if(rBLAST::has_blast()) {
    # 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 <- clustirr(s = rbind(a, b))

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

    # get honeycombs
    g <- get_honeycombs(com = gcd$community_occupancy_matrix)
    g
}
```

get_nrads *Compute Normalized Rank Abundance Distributions (NRADs)*

Description

This function NRADs for a given community occupancy matrix. It performs normalization using the RADnormalization_matrix function implemented in RADanalysis.

Usage

```
get_nrads(community_occupancy_matrix, B = 1000)
```

Arguments

community_occupancy_matrix	A matrix where rows represent communities and columns represent samples (generated by detect_communities). Entries are the abundances of communities in each sample (column).
B	A positive integer specifying the number of iterations for averaging during normalization. Default is 1000.

Details

The function computes the NRADs by ranking the community abundances in each sample (repertoire), determining the maximum rank, and normalizing the abundances using the RADnormalization_matrix function. The result is returned as a melted data frame containing the normalized abundances for each sample and rank. This enables comparison of community structures across repertoires with different depths (numbers of clonotypes).

The parameter B specifies the number of bootstrap iterations used in the normalization process. A higher value of B increases the stability and reliability of the normalized abundances but also increases computation time. The default value is 1000, which is a common choice for balancing accuracy and computational efficiency.

Value

A data frame containing the normalized abundances for each sample and rank. The data frame has the following columns:

- sample: The sample identifier.
- rank: The rank of the abundance.
- norm.abundance: The normalized abundance at the given rank.

Examples

```
# Example usage:
# Create a sample community occupancy matrix
community_occupancy_matrix <- matrix(rpois(100, lambda = 10),nrow=20,ncol=5)

# Compute normalized RADs
nrads <- get_nrads(community_occupancy_matrix, B = 100)

# Display the first few rows of the result
head(nrads)
```

`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 `clustirr`.

Usage

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

Arguments

`g`	Object returned by the function `clustirr`
`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. 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

```
# only run if blast is installed
if(rBLAST::has_blast()) {
    # load package input data
    data("CDR3ab", package = "ClustIRR")
    s <- data.frame(CDR3b = CDR3ab[1:100, "CDR3b"], sample = "A", clone_size = 1)

    # run ClustIRR analysis
    c <- clustirr(s = s)

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

`save_interactive_graph`

Save interactive ClustIRR graph

Description

This function saves an interactive `visNetwork` object as html file of a `clustirr` graph.

Usage

```
save_interactive_graph(graph,
            file_name,
            output_folder,
            overwrite = TRUE)
```

Arguments

`graph`	Object returned by the function `plot_graph`.
`file_name`	Name of the exported graph.
`output_folder`	Path, where to store the graph.
`overwrite`	Whether to overwrite existing files.

Value

This function saves an interactive `visNetwork` plot as self-contained html file. If `output_folder` is not existent, it will be saved in the working directory.

Examples

```
# only run if blast is installed
if(rBLAST::has_blast()) {
    # load package input data
    data("CDR3ab", package = "ClustIRR")
    s <- data.frame(CDR3b = CDR3ab[1:100, "CDR3b"], sample = "A", clone_size = 1)

    # run ClustIRR analysis
    c <- clustirr(s = s)

    # plot graph with vertices as clones (has to be a visNetwork)
    g <- plot_graph(c, as_visnet=TRUE, show_singletons=TRUE, node_opacity = 0.8)

    # # Save the graph to a temporary directory
    # my_temp_dir <- tempdir()
    # save_interactive_graph(graph = g,
    #                    file_name = "test_graph",
    #                    output_folder = my_temp_dir,
    #                    overwrite = TRUE)
}
```

---

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, December 2024

Examples

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

Index