Package 'ClustIRR'

July 16, 2025

Type Package

Title Clustering of immune receptor repertoires

Version 1.7.1

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 blaster, future, future.apply, 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

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

SystemRequirements GNU make

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)

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

git_branch devel

git_last_commit 5930b5b

git_last_commit_date 2025-04-15

Repository Bioconductor 3.22

Date/Publication 2025-07-16

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

Maintainer Simo Kitanovski <simokitanovski@gmail.com>

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
```

---

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,
         cores = 1,
         control = list(gmi = 0.7,
                        trim_flank_aa = 3,
                        db_dist = 0,
                        db_custom = NULL))
```

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.

`cores` number of computer cores to use (default = 1) `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 `db_dist` (default = 0), then we annotate the CDR3 from `s` with the specificity of the database CDR3 sequence.

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

```
# 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)
```

---

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 <- 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)
```

Datasets *Datasets* CDR3ab *and* D1 *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.

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 as stores as element of a list. For each TCR repertoires we have a metadata: ma, mb, and mc.

Usage

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

# For dataset D1
data(D1)
```

Format

`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")
```

`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

```
# 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,
                          weight = "ncweight",
                          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_communities *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.

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.

Usage

```
decode_communities(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.

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

```
# 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,
                          weight = "ncweight",
                          iterations = 100,
                          chains = c("CDR3a", "CDR3b"))

# We "decompose" the communities in the gcd object using decode_community
# based on the 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 based on 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_communities(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,
                   weight = "nweight",
                   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.
`weight`	which edge weight attribute (default = nweight) should be used 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

```
# 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,
                          weight = "ncweight",
                          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$community_summary$wide)

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

`get_ag_summary` *Estimate the number of antigen-specific T-cells in selected communities*

Description

Use `node_summary` data.frame generated by the function `detect_communities`; and 2. antigen species/genes to estimate the number of antigen-specific T-cells in selected communities in each repertoire.

Usage

```
get_ag_summary(node_summary,
               ag_species,
               ag_genes,
               db = "vdjdb",
               db_dist = 0,
               chain = "both")
```

Arguments

`node_summary`	`node_summary` data.frame
`ag_species`	antigen species, character vector, e.g. c("EBV", "CMV")
`ag_genes`	antigen genes, character vector, e.g. "MLANA"
`db`	annotation database, character, e.g. "vdjdb"
`db_dist`	maximum edit distance threshold for matching, nummeric
`chain`	immune receptor chain for annotation, "both", "CDR3a" or "CDR3b"

Details

The user has to provide a vector of antigen species (e.g. `ag_species` = c("EBV", "CMV")) and/or a vector of antigen genes (e.g. `ag_genes` = "MLANA"). Furthermore, the user has to provide nodes (`node_summary` data.frame created by the function `detect_communities`) and a vector with community IDs.

The user can also select an annotation database `db`, such as "vdjdb", "mcpas" or "tcr3d"; and restrict the annotation to specific IR chains, such as "CDR3a", "CDR3b" or "both". By default, we will look for perfect matches (`db_dist=0`) between CDR3 sequences in the input and in the annotation database for annotation. Flexible annotation based on edit distances can be performed by increasing `db_dist`.

Value

The output is a data.frame with the number of T-cells specific for the antigenic species/genes (columns) provided as input per repertoire (row), including the total number of T-cells in each repertoire.

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 <- clustirr(s = rbind(a, b))

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

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

ag_summary <- get_ag_summary(node_summary = gcd$node_summary,
                             ag_species = c("EBV", "CMV"),
                             ag_genes = "MLANA",
                             db = "vdjdb",
                             db_dist = 0,
                             chain = "both")
```

---

get_beta_scatterplot *Compare community βs between pairs of repertoires*

Description

Visualize the β means as a 2D scatterplot, representing relative community occupancies for all pairs of repertoires. At the same time, annotate the communities (dots) based on their specificity.

Usage

```
get_beta_scatterplot(beta,
                     node_summary,
                     ag_species,
                     ag_genes,
                     db = "vdjdb",
                     db_dist = 0,
                     chain = "both")
```

Arguments

beta	beta data.frame
node_summary	node_summary data.frame
ag_species	antigen species, character vector, e.g. c("EBV", "CMV")
ag_genes	antigen genes, character vector, e.g. "MLANA"
db	annotation database, character, e.g. "vdjdb"
db_dist	maximum edit distance threshold for matching, nummeric
chain	immune receptor chain for annotation, "both", "CDR3a" or "CDR3b"

Details

The user has to provide a vector of antigen species (e.g. ag_species = c("EBV", "CMV")) and/or a vector of antigen genes (e.g. ag_genes = "MLANA"). 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"; and restrict the annotation to specific IR chains, such as "CDR3a", "CDR3b" or "both". By default, we will look for perfect matches (db_dist=0) between CDR3 sequences in the input and in the annotation database for annotation. Flexible annotation based on edit distances can be performed by increasing db_dist.

Value

The output is a list with 4 elements:

node_annotations: annotated node_summary

beta_summary: annotated beta

vars: annotation variables

scatterplots: a list of scatterplots: Each element of the list contains a scatterplots for a specific antigen species/gene. Within each element of the list there are n^2 panels (comparisons between repertoire pairs), with n as the number of repertoires.

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 <- clustirr(s = rbind(a, b))

# detect communities
gcd <- detect_communities(graph = c$graph,
```

```
                              algorithm = "leiden",
                              resolution = 1,
                              weight = "ncweight",
                              chains = c("CDR3a", "CDR3b"))

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

# generate beta violin plots
beta_scatterplot <- get_beta_scatterplot(beta = dco$posterior_summary$beta,
                                         node_summary = gcd$node_summary,
                                         ag_species = c("EBV", "CMV"),
                                         ag_genes = "MLANA",
                                         db = "vdjdb",
                                         db_dist = 0,
                                         chain = "both")
```

---

get_beta_violins *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_violins(beta,
                 node_summary,
                 ag_species,
                 ag_genes,
                 db = "vdjdb",
                 db_dist = 0,
                 chain = "both")
```

Arguments

`beta`	`beta` data.frame
`node_summary`	`node_summary` data.frame
`ag_species`	antigen species, character vector, e.g. c("EBV", "CMV")
`ag_genes`	antigen genes, character vector, e.g. "MLANA"
`db`	annotation database, character, e.g. "vdjdb"
`db_dist`	maximum edit distance threshold for matching, nummeric
`chain`	immune receptor chain for annotation, "both", "CDR3a" or "CDR3b"

Details

The user has to provide a vector of antigen species (e.g. `ag_species` = c("EBV", "CMV")) and/or a vector of antigen genes (e.g. `ag_genes` = "MLANA"). 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"; and restrict the annotation to specific IR chains, such as "CDR3a", "CDR3b" or "both". By default, we will look for perfect matches (`db_dist=0`) between CDR3 sequences in the input and in the annotation database for annotation. Flexible annotation based on edit distances can be performed by increasing `db_dist`.

Value

The output is a list with 4 elements:

`node_annotations`: annotated `node_summary`

`beta_summary`: annotated `beta`

`vars`: annotation variables

`violins`: violin plots (one for each antigen species and gene)

`violins`: a list of violin plots. Each element of the list contains a violin visual for a specific antigen species/gene.

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 <- clustirr(s = rbind(a, b))

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

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

# generate beta violin plots
beta_violins <- get_beta_violins(beta = dco$posterior_summary$beta,
                                 node_summary = gcd$node_summary,
                                 ag_species = c("EBV", "CMV"),
                                 ag_genes = "MLANA",
                                 db = "vdjdb",
                                 db_dist = 0,
                                 chain = "both")
```

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

```
# 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,
                          weight = "ncweight",
                          chains = c("CDR3a", "CDR3b"))

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

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 `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

```
# 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)
```

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