Package 'immunarch'

December 15, 2022

Type Package

Title Bioinformatics Analysis of T-Cell and B-Cell Immune Repertoires

Version 0.9.0

Contact support@immunomind.io

Description
A comprehensive framework for bioinformatics exploratory analysis of bulk and single-cell T-cell receptor and antibody repertoires. It provides seamless data loading, analysis and visualisation for AIRR (Adaptive Immune Receptor Repertoire) data, both bulk immunosequencing (RepSeq)
and single-cell sequencing (scRNAseq). Immunarch implements most of the widely used AIRR analysis methods,
such as: clonality analysis, estimation of repertoire similarities in distribution of clonotypes and gene segments, repertoire diversity analysis, annotation of clonotypes using external immune receptor
databases and clonotype tracking in vaccination and cancer studies. A successor to our previously published 'tcR' immunoinformatics package (Nazarov 2015) <doi:10.1186/s12859-015-0613-1>.

License AGPL-3

URL https://immunarch.com/, https://github.com/immunomind/immunarch

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

Imports factoextra (>= 1.0.4), fpc, UpSetR (>= 1.4.0), pheatmap (>= 1.0.12), ggrepel (>= 0.8.0), reshape2 (>= 1.4.2), circlize, MASS (>= 7.3), Rtsne (>= 0.15), readxl (>= 1.3.1), shiny (>= 1.4.0), shinythemes, airr, ggseqlogo, ggalluvial (>= 0.10.0), Rcpp (>= 1.0), magrittr, methods, scales, ggpubr (>= 0.2), rlang (>= 0.4), plyr, purrr, stringdist, jsonlite, readr, stringr, tibble, tidyselect, tidyr, igraph, ape, doParallel, rlist, utils, glue, phangorn, uuid, stringi, ggraph

Depends R (>= 4.0.0), ggplot2 (>= 3.1.0), dplyr (>= 0.8.0), dtplyr (>= 1.0.0), data.table (>= 1.12.6), patchwork

LinkingTo Rcpp

Suggests knitr (>= 1.8), roxygen2 (>= 3.0.0), testthat (>= 2.1.0), pkgdown (>= 0.1.0), assertthat, rmarkdown

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.2.2

LazyData true

LazyDataCompression xz

NeedsCompilation yes

Author Vadim I. Nazarov [aut, cre],
Vasily O. Tsvetkov [aut],
Eugene Rumynskiy [aut],
Aleksandr A. Popov [aut],
Ivan Balashov [aut],
Maria Samokhina [aut],
Anna Lorenc [ctb],
Daniel J. Moore [ctb],
Victor Greiff [ctb],
ImmunoMind [cph, fnd]

Maintainer Vadim I. Nazarov `<support@immunomind.io>`

Repository CRAN

Date/Publication 2022-12-15 15:00:05 UTC

R topics documented:

`.quant_column_choice` *Get a column's name using the input alias*

Description

Get a column's name using the input alias

Usage

```
.quant_column_choice(x)
```

Arguments

`x`	Character vector of length 1.

Value

A string with the column name.

Developer Examples

immunarch:::.quant_column_choice("count") immunarch:::.quant_column_choice("freq")

`aa_properties` *Tables with amino acid properties*

Description

Tables with amino acid properties

`aa_table` *Amino acid / codon table*

Description

Amino acid / codon table

Usage

```
AA_TABLE
```

Format

An object of class `table` of length 65.

`add_class` *Add a new class attribute*

Description

Add a new class attribute

Usage

```
add_class(.obj, .class)
```

Arguments

`.obj`	R object.
`.class`	String with the desired class name.

Value

Input object with additional class `.class`.

Developer Examples

tmp <- "abc" class(tmp) tmp <- immunarch:::add_class(tmp, "new_class") class(tmp)

apply_symm *Apply function to each pair of data frames from a list.*

Description

Apply the given function to every pair in the given datalist. Function either symmetrical (i.e. fun(x,y) == fun(y,x)) or assymmetrical (i.e. fun(x,y) != fun(y,x)).

Usage

```
apply_symm(.datalist, .fun, ..., .diag = NA, .verbose = TRUE)

apply_asymm(.datalist, .fun, ..., .diag = NA, .verbose = TRUE)
```

Arguments

`.datalist`	List with some data.frames.
`.fun`	Function to apply, which return basic class value.
`...`	Arguments passsed to .fun.
`.diag`	Either NA for NA or something else != NULL for .fun(x,x).
`.verbose`	if TRUE then output a progress bar.

Value

Matrix with values M[i,j] = fun(datalist[i], datalist[j])

Examples

```
data(immdata)
apply_symm(immdata$data, function(x, y) {
  nrow(x) + nrow(y)
})
```

bcrdata *BCR dataset*

Description

A dataset with BCR data for testing and examplatory purposes.

Usage

```
bcrdata
```

Format

A list of two elements. The first element ("data") is a list of 1 element named "full_clones" that contains immune repertoire data frame. The second element ("meta") is empty metadata table.

data List of immune repertoire data frames.

meta Metadata ...

`bunch_translate` *Nucleotide to amino acid sequence translation*

Description

Nucleotide to amino acid sequence translation

Usage

```
bunch_translate(.seq, .two.way = TRUE, .ignore.n = FALSE)
```

Arguments

`.seq`	Vector or list of strings.
`.two.way`	Logical. If TRUE (default) then translate from the both ends (like MIXCR).
`.ignore.n`	Logical. If FALSE (default) then return NA for sequences that have N, else parse triplets with N as ~

Value

Character vector of translated input sequences.

Examples

```
data(immdata)
head(bunch_translate(immdata$data[[1]]$CDR3.nt))
```

`check_distribution` *Check and normalise distributions*

Description

Check if the given .data is a distribution and normalise it if necessary with an optional Laplace correction.

Usage

```
check_distribution(
  .data,
  .do.norm = NA,
  .laplace = 1,
  .na.val = 0,
  .warn.zero = FALSE,
  .warn.sum = TRUE
)
```

Arguments

`.data`	Numeric vector of values.
`.do.norm`	One of the three values - NA, TRUE or FALSE. If NA then checks for distrubution (sum(.data) == 1) and normalises if needed with the given laplace correction value. if TRUE then does the normalisation and laplace correction. If FALSE then doesn't do either normalisaton or laplace correction.
`.laplace`	Value for the laplace correction.
`.na.val`	Replace all NAs with this value.
`.warn.zero`	if TRUE then the function checks if in the resulted vector (after normalisation) are any zeros, and prints a warning message if there are some.
`.warn.sum`	if TRUE then the function checks if the sum of resulted vector (after normalisation) is equal to one, and prints a warning message if not.

Value

Numeric vector.

Developer Examples

immunarch:::check_distribution(c(1, 2, 3)) immunarch:::check_distribution(c(1, 2, 3), TRUE) immunarch:::check_distribution(c(1, 2, 3), FALSE)

coding — *Filter out coding and non-coding clonotype sequences*

Description

Filter out clonotypes with non-coding, coding, in-frame or out-of-frame CDR3 sequences:

‘coding()‘ - remove all non-coding sequences (i.e., remove all sequences with stop codons and frame shifts);

‘noncoding()‘ - remove all coding sequences (i.e., leave sequences with stop codons and frame shifts only);

‘inframes()‘ - remove all out-of-frame sequences (i.e., remove all sequences with frame shifts);

‘outofframes()‘ - remove all in-frame sequences (i.e., leave sequences with frame shifts only).

Note: the function will remove all clonotypes sequences with NAs in the CDR3 amino acid column.

Usage

```
coding(.data)

noncoding(.data)

inframes(.data)

outofframes(.data)
```

Arguments

`.data` The data to be processed. Can be data.frame, data.table, or a list of these objects.

Every object must have columns in the immunarch compatible format. immunarch_data_format

Competent users may provide advanced data representations: DBI database connections, Apache Spark DataFrame from copy_to or a list of these objects. They are supported with the same limitations as basic objects.

Note: each connection must represent a separate repertoire.

Value

Filtered data frame.

Examples

```
data(immdata)
immdata_cod <- coding(immdata$data)
immdata_cod1 <- coding(immdata$data[[1]])
```

dbAnnotate *Annotate clonotypes in immune repertoires using clonotype databases such as VDJDB and MCPAS*

Description

Annotate clonotypes using immune receptor databases with known condition-associated receptors. Before using this function, you need to download database files first. For more details see the tutorial https://immunarch.com/articles/web_only/v11_db.html.

Usage

```
dbAnnotate(.data, .db, .data.col, .db.col)
```

Arguments

`.data` The data to process. It can be a data.frame, a data.table, or a list of these objects.

Every object must have columns in the immunarch compatible format. immunarch_data_format

Competent users may provide advanced data representations: DBI database connections, Apache Spark DataFrame from copy_to or a list of these objects. They are supported with the same limitations as basic objects.

Note: each connection must represent a separate repertoire.

`.db` A data frame or a data table with an immune receptor database. See dbLoad on how to load databases into R.

`.data.col` Character vector. Vector of columns in the input repertoires to use for clonotype search. E.g., ‘"CDR3.aa"‘ or ‘c("CDR3.aa", "V.name")‘.

`.db.col` Character vector. Vector of columns in the database to use for clonotype search. The order must match the order of ".data.col". E.g., if ".data.col" is ‘c("CDR3.aa", "V.name")‘, then ".db.col" must have the exact order of columns. i.e., the first column must correspond to CDR3 amino acid sequences, and the second column must correspond to V gene segment names.

Value

Data frame with input sequences and counts or proportions for each of the input repertoire.

Examples

```
data(immdata)

#' # Example file path
file_path <- paste0(system.file(package = "immunarch"), "/extdata/db/vdjdb.example.txt")

# Load the database with human-only TRB-only receptors for all known antigens
db <- dbLoad(file_path, "vdjdb", "HomoSapiens", "TRB")
```

```
res <- dbAnnotate(immdata$data, db, "CDR3.aa", "cdr3")
res
```

dbLoad — *Load clonotype databases such as VDJDB and McPAS into the R workspace*

Description

The function automatically detects the database format and loads it into R. Additionally, the function provides a general query interface to databases that allows filtering by species, chain types (i.e., locus) and pathology (i.e., antigen species).

Currently we support three popular databases:

VDJDB - https://github.com/antigenomics/vdjdb-db

McPAS-TCR - http://friedmanlab.weizmann.ac.il/McPAS-TCR/

TBAdb from PIRD - https://db.cngb.org/pird/

Usage

```
dbLoad(.path, .db, .species = NA, .chain = NA, .pathology = NA)
```

Arguments

`.path`	Character. A path to the database file, e.g., "/Users/researcher/Downloads/McPAS-TCR.csv".
`.db`	Character. A database type: either "vdjdb", "vdjdb-search", "mcpas" or "tbadb". "vdjdb" for VDJDB; "vdjdb-search" for search table obtained from the web interface of VDJDB; "mcpas" for McPAS-TCR; "tbadb" for PIRD TBAdb.
`.species`	Character. A string or a vector of strings specifying which species need to be in the database, e.g., "HomoSapiens". Pass NA (by default) to load all available species.
`.chain`	Character. A string or a vector of strings specifying which chains need to be in the database, e.g., "TRB". Pass NA (by default) to load all available chains.
`.pathology`	Character. A string or a vector of strings specifying which disease, virus, bacteria or any condition needs to be in the database, e.g., "CMV". Pass NA (by default) to load all available conditions.

Value

Data frame with the input database records.

Examples

```
# Example file path
file_path <- paste0(system.file(package = "immunarch"), "/extdata/db/vdjdb.example.txt")

# Load the database with human-only TRB-only receptors for all known antigens
db <- dbLoad(file_path, "vdjdb", "HomoSapiens", "TRB")
db
```

entropy *Information measures*

Description

Compute information-based estimates and distances.

Usage

```
entropy(.data, .base = 2, .norm = FALSE, .do.norm = NA, .laplace = 1e-12)

kl_div(.alpha, .beta, .base = 2, .do.norm = NA, .laplace = 1e-12)

js_div(.alpha, .beta, .base = 2, .do.norm = NA, .laplace = 1e-12, .norm.entropy = FALSE)

cross_entropy(.alpha, .beta, .base = 2, .do.norm = NA,
              .laplace = 1e-12, .norm.entropy = FALSE)
```

Arguments

`.data` Numeric vector. Any distribution.

`.base` Numeric. A base of logarithm.

`.norm` Logical. If TRUE then normalises the entropy by the maximal value of the entropy.

`.do.norm` If TRUE then normalises the input distributions to make them sum up to 1.

`.laplace` Numeric. A value for the laplace correction.

`.alpha` Numeric vector. A distribution of some random value.

`.beta` Numeric vector. A distribution of some random value.

`.norm.entropy` Logical. If TRUE then normalises the resulting value by the average entropy of input distributions.

Value

A numeric value.

Examples

```
P <- abs(rnorm(10))
Q <- abs(rnorm(10))
entropy(P)
kl_div(P, Q)
js_div(P, Q)
cross_entropy(P, Q)
```

---

fixVis *Manipulate ggplot plots and create publication-ready plots*

Description

The fixVis is a built-in software tool for the manipulation of plots, such as adjusting title text font and size, axes, and more. It is a powerful tool designed to produce publication-ready plots with minimal amount of coding.

Usage

```
fixVis(.plot = NA)
```

Arguments

.plot	A ggplot2 plot.

Value

No return value because it is an application.

Examples

```
if (interactive()) {
  # Compute gene usage, visualise it and tweak via fixVis
  data(immdata) # load test data
  gu <- geneUsage(immdata$data)
  p <- vis(gu)
  fixVis(p)
}
```

geneUsage *Main function for estimation of V-gene and J-gene statistics*

Description

An utility function to analyse the immune receptor gene usage (IGHD, IGHJ, IDHV, IGIJ, IGKJ, IGKV, IGLJ, IGLV, TRAJ, TRAV, TRBD, etc.) and statistics. For gene details run `gene_stats()`.

Usage

```
geneUsage(
  .data,
  .gene = c("hs.trbv", "HomoSapiens.TRBJ", "macmul.IGHV"),
  .quant = c(NA, "count"),
  .ambig = c("inc", "exc", "maj"),
  .type = c("segment", "allele", "family"),
  .norm = FALSE
)
```

Arguments

`.data` The data to be processed. Can be data.frame, data.table, or a list of these objects.

Every object must have columns in the immunarch compatible format. immunarch_data_format

Competent users may provide advanced data representations: DBI database connections, Apache Spark DataFrame from copy_to or a list of these objects. They are supported with the same limitations as basic objects.

Note: each connection must represent a separate repertoire.

`.gene` A character vector of length one with the name of the gene you want to analyse of the specific species. If you provide a vector of different length, only the first element will be used. The string should also contain the species of interest, for example, valid ".gene" arguments are "hs.trbv", "HomoSapiens.TRBJ" or "macmul.IGHV". For details run `gene_stats()`.

`.quant` Selects the column with data to evaluate. Pass NA if you want to compute gene statistics at the clonotype level without re-weighting. Pass "count" to use the "Clones" column to weight genes by abundance of their corresponding clonotypes.

`.ambig` An option to handle ambiguous gene assigments, e.g., "TRAV1,TRAV2".

- Pass "inc" to include all possible gene segments, so "TRAV1,TRAV2" is counted as a different gene segment.

- Pass "exc" to exclude all ambiguous gene assignments, so "TRAV1,TRAV2" is excluded from the resultant gene table.

We recommend to turn it on by passing "inc" (turned on by default). You can exclude data for the cases where there is no clear match for gene, include it for every supplied gene, or pick only first from the set. Set it to "exc", "inc" or "maj", respectively.

.type Set the type of data to evaluate: "segment", "allele", or "family".

.norm If TRUE then return proportions of genes. If FALSE then return counts of genes.

Value

A data frame with rows corresponding to gene segments and columns corresponding to the input samples.

Examples

```
data(immdata)
gu <- geneUsage(immdata$data)
vis(gu)
```

geneUsageAnalysis *Post-analysis of V-gene and J-gene statistics: PCA, clustering, etc.*

Description

The geneUsageAnalysis function deploys several data analysis methods, including PCA, multi-dimensional scaling, Jensen-Shannon divergence, k-means, hierarchical clustering, DBscan, and different correlation coefficients.

Usage

```
geneUsageAnalysis(
  .data,
  .method = c("js+hclust", "pca+kmeans", "anova", "js+pca+kmeans"),
  .base = 2,
  .norm.entropy = FALSE,
  .cor = c("pearson", "kendall", "spearman"),
  .do.norm = TRUE,
  .laplace = 1e-12,
  .verbose = TRUE,
  .k = 2,
  .eps = 0.01,
  .perp = 1,
  .theta = 0.1
)
```

Arguments

.data The geneUsageAnalysis function runs on the output from geneUsage.

.method A string that defines the type of analysis to perform. Can be "pca", "mds", "js", "kmeans", "hclust", "dbscan" or "cor" if you want to calculate correlation coefficient. In the latter case you have to provide .cor argument.

.base A numerical value that defines the logarithm base for Jensen-Shannon divergence.

.norm.entropy A logical value. Set TRUE to normalise your data if you haven't done it already.

.cor A string that defines the correlation coefficient for analysis. Can be "pearson", "kendall" or "spearman".

.do.norm A logical value. If TRUE it forces Laplace smoothing, if NA it checks if smoothing is necessary, if FALSE does nothing.

.laplace The numeric value, which is used as a pseudocount for Laplace smoothing.

.verbose A logical value.

.k The number of clusters to create, passed as k to hcut or as centers to kmeans.

.eps A numerical value, DBscan epsylon parameter, see immunr_dbscan.

.perp A numerical value, t-SNE perplexity, see immunr_tsne.

.theta A numerical value, t-SNE theta parameter, see immunr_tsne.

Value

Depends on the last element in the .method string. See immunr_tsne for more info.

Examples

```
data(immdata)
gu <- geneUsage(immdata$data, .norm = TRUE)
geneUsageAnalysis(gu, "js+hclust", .verbose = FALSE) %>% vis()
```

gene_segments *Gene segments table*

Description

Gene segments table

gene_stats *WIP*

Description

WIP

Usage

```
gene_stats()
```

Value

`gene_stats` returns all segment gene statistics

Examples

```
gene_stats()
get_genes("hs.trbv", "segment")
```

getKmers *Calculate the k-mer statistics of immune repertoires*

Description

Calculate the k-mer statistics of immune repertoires

Usage

```
getKmers(.data, .k, .col = c("aa", "nt"), .coding = TRUE)
```

Arguments

`.data` The data to be processed. Can be data.frame, data.table, or a list of these objects.

Every object must have columns in the immunarch compatible format. immunarch_data_format

Competent users may provide advanced data representations: DBI database connections, Apache Spark DataFrame from copy_to or a list of these objects. They are supported with the same limitations as basic objects.

Note: each connection must represent a separate repertoire.

`.k` Integer. Length of k-mers.

`.col` Character. Which column to use, pass "aa" (by default) for CDR3 amino acid sequence, pass "nt" for CDR3 nucleotide sequences.

`.coding` Logical. If TRUE (by default) then removes all non-coding sequences from input data first.

Value

Data frame with two columns (k-mers and their counts).

Examples

```
data(immdata)
kmers <- getKmers(immdata$data[[1]], 5)
kmers %>% vis()
```

`group_from_metadata` *Get a character vector of samples' groups from the input metadata file*

Description

Get a character vector of samples' groups from the input metadata file

Usage

```
group_from_metadata(.by, .metadata, .sep = "; ")
```

Arguments

`.by` Character vector. Specify a column or columns in the input metadata to group by.

`.metadata` Metadata object.

`.sep` Character vector. Defines a separator between groups if more than one group passed in `.by`.

Value

Character vector with group names.

Developer Examples

immunarch:::group_from_metadata("Status", data.frame(Status = c("A", "A", "B", "B", "C")))

`has_class` *Check for the specific class*

Description

A function to check if an input object has a specific class name.

Usage

```
has_class(.data, .class)
```

Arguments

`.data` Any R object.

`.class` Character vector. Specifies a class name to check against.

Value

Logical value.

Developer Examples

tmp <- "abc" immunarch:::has_class(tmp, "new_class") tmp <- immunarch:::add_class(tmp, "new_class") immunarch:::has_class(tmp, "new_class")

immdata *Single chain immune repertoire dataset*

Description

A dataset with single chain TCR data for testing and examplatory purposes.

Usage

```
immdata
```

Format

A list of two elements. The first element ("data") is a list with data frames with clonotype tables. The second element ("meta") is a metadata table.

data List of immune repertoire data frames.

meta Metadata ...

immunr_data_format *Specification of the data format used by immunarch dataframes*

Description

- "Clones" - number of barcodes (events, UMIs) or reads;

- "Proportion" - proportion of barcodes (events, UMIs) or reads;

- "CDR3.nt" - CDR3 nucleotide sequence;

- "CDR3.aa" - CDR3 amino acid sequence;

- "V.name" - names of aligned Variable gene segments;

- "D.name" - names of aligned Diversity gene segments or NA;

- "J.name" - names of aligned Joining gene segments;

- "V.end" - last positions of aligned V gene segments (1-based);

- "D.start" - positions of D’5 end of aligned D gene segments (1-based);

- "D.end" - positions of D’3 end of aligned D gene segments (1-based);

- "J.start" - first positions of aligned J gene segments (1-based);

- "VJ.ins" - number of inserted nucleotides (N-nucleotides) at V-J junction (-1 for receptors with VDJ recombination);

- "VD.ins" - number of inserted nucleotides (N-nucleotides) at V-D junction (-1 for receptors with VJ recombination);

- "DJ.ins" - number of inserted nucleotides (N-nucleotides) at D-J junction (-1 for receptors with VJ recombination);

- "Sequence" - full nucleotide sequence.

immunr_hclust *Clustering of objects or distance matrices*

Description

Clusters the data with one of the following methods:

- immunr_hclust clusters the data using the hierarchical clustering from hcut;

- immunr_kmeans clusters the data using the K-means algorithm from kmeans;

- immunr_dbscan clusters the data using the DBSCAN algorithm from dbscan.

Usage

```
immunr_hclust(.data, .k = 2, .k.max = nrow(.data) - 1, .method = "complete", .dist = TRUE)

immunr_kmeans(.data, .k = 2, .k.max = as.integer(sqrt(nrow(.data))) + 1,
.method = c("silhouette", "gap_stat"))

immunr_dbscan(.data, .eps, .dist = TRUE)
```

Arguments

.data	Matrix or data frame with features, distance matrix or output from repOverlapAnalysis or geneUsageAnalysis functions.
.k	The number of clusters to create, defined as k to hcut or as centers to kmeans.
.k.max	Limits the maximum number of clusters. It is passed as k.max to fviz_nbclust for immunr_hclust and immunr_kmeans.
.method	Passed to hcut or as fviz_nbclust. In case of hcut the agglomeration method is going to be used (argument hc_method). In case of fviz_nbclust it is the method to be used for estimating the optimal number of clusters (argument method).
.dist	If TRUE then ".data" is expected to be a distance matrix. If FALSE then the euclidean distance is computed for the input objects.
.eps	Local radius for expanding clusters, minimal distance between points to expand clusters. Passed as eps to dbscan.

Value

`immunr_hclust` - list with two elements. The first element is an output from hcut. The second element is an output from fviz_nbclust

`immunr_kmeans` - list with three elements. The first element is an output from kmeans. The second element is an output from fviz_nbclust. The third element is the input dataset `.data`.

`immunr_dbscan` - list with two elements. The first element is an output from dbscan. The second element is the input dataset `.data`.

Examples

```
data(immdata)
gu <- geneUsage(immdata$data, .norm = TRUE)
immunr_hclust(t(as.matrix(gu[, -1])), .dist = FALSE)

gu[is.na(gu)] <- 0
immunr_kmeans(t(as.matrix(gu[, -1])))
```

`immunr_pca` *Dimensionality reduction*

Description

Collects a set of principal variables, reducing the number of not important variables to analyse. Dimensionality reduction makes data analysis algorithms work faster and sometimes more accurate, since it also reduces noise in the data. Currently available methods are:

- `immunr_pca` performs PCA (Principal Component Analysis) using prcomp;

- `immunr_mds` performs MDS (Multi-Dimensional Scaling) using isoMDS;

- `immunr_tsne` performs tSNE (t-Distributed Stochastic Neighbour Embedding) using Rtsne.

Usage

```
immunr_pca(.data, .scale = default_scale_fun, .raw = TRUE, .orig = FALSE, .dist = FALSE)

immunr_mds(.data, .scale = default_scale_fun, .raw = TRUE, .orig = FALSE, .dist = TRUE)

immunr_tsne(.data, .perp = 1, .dist = TRUE, ...)
```

Arguments

`.data` A matrix or a data frame with features, distance matrix or output from repOverlapAnalysis or geneUsageAnalysis functions.

`.scale` A function to apply to your data before passing it to any of dimensionality reduction algorithms. There is no scaling by default.

`.raw` If TRUE then returns the non-processed output from dimensionality reduction algorithms. Pass FALSE if you want to visualise results.

`.orig` If TRUE then returns the original result from algorithms. Pass FALSE if you want to visualise results.

`.dist` If TRUE then assumes that ".data" is a distance matrix.

`.perp` The perplexity parameter for Rtsne. Sepcifies the number of neighbours each data point must have in the resulting plot.

`...` Other parameters passed to Rtsne.

Value

`immunr_pca` - an output from prcomp.

`immunr_mds` - an output from isoMDS.

`immunr_tsne` - an output from Rtsne.

See Also

vis.immunr_pca for visualisations.

Examples

```
data(immdata)
gu <- geneUsage(immdata$data)
gu[is.na(gu)] <- 0
gu <- t(as.matrix(gu[, -1]))
immunr_pca(gu)
immunr_mds(dist(gu))
immunr_tsne(dist(gu))
```

`inc_overlap` *Incremental counting of repertoire similarity*

Description

For reference please look up https://www.pnas.org/content/111/16/5980 (Fig. 4).

Usage

```
inc_overlap(
  .data,
  .fun,
  .step = 1000,
  .n.steps = 10,
  .downsample = FALSE,
  .bootstrap = NA,
  .verbose.inc = TRUE,
  ...
)
```

Arguments

`.data` The data to be processed. Can be data.frame, data.table, or a list of these objects.

Every object must have columns in the immunarch compatible format. immunarch_data_format

Competent users may provide advanced data representations: DBI database connections, Apache Spark DataFrame from copy_to or a list of these objects. They are supported with the same limitations as basic objects.

Note: each connection must represent a separate repertoire.

`.fun` Function to compute overlaps. e.g., `morisita_index`.

`.step` Either an integer or a numeric vector.

In the first case, the integer defines the step of incremental overlap.

In the second case, the vector encodes all repertoire sampling depths.

`.n.steps` Integer. Number of steps if `.step` is a single integer. Skipped if ".step" is a numeric vector.

`.downsample` If TRUE then performs downsampling to N clonotypes at each step instead of choosing the top N clonotypes.

`.bootstrap` Set NA to turn off any bootstrapping, set a number to perform bootstrapping with this number of tries.

`.verbose.inc` Logical. If TRUE then shows the output from the computation process.

`...` Other arguments passed to `.fun`.

Value

List with overlap matrices.

Examples

```
data(immdata)
ov <- repOverlap(immdata$data, "inc+overlap", .step = 100, .verbose.inc = FALSE, .verbose = FALSE)
vis(ov)
```

`matrixdiagcopy` *Copy the upper matrix triangle to the lower one*

Description

Copy the upper matrix triangle to the lower one

Usage

```
matrixdiagcopy(.mat)
```

Arguments

`.mat` Matrix.

Value

Matrix with its upper tri part copied to the lower tri part.

Developer Examples

mat <- matrix(0, 3, 3) mat mat[1, 3] <- 1 mat <- immunarch:::matrixdiagcopy(mat) mat

public_matrix *Get a matrix with public clonotype frequencies*

Description

Get a matrix with public clonotype frequencies

Usage

```
public_matrix(.data)
```

Arguments

`.data`	Public repertoire, an output from pubRep.

Value

Matrix with per-sample clonotype counts / proportions only.

Examples

```
data(immdata)
immdata$data <- lapply(immdata$data, head, 2000)
pr <- pubRep(immdata$data, .verbose = FALSE)
pr.mat <- public_matrix(pr)
dim(pr.mat)
head(pr.mat)
```

pubRep *Create a repertoire of public clonotypes*

Description

Create a repertoire of public clonotypes

Usage

```
pubRep(
  .data,
  .col = "aa+v",
  .quant = c("count", "prop"),
  .coding = TRUE,
  .min.samples = 1,
  .max.samples = NA,
  .verbose = TRUE
)
```

Arguments

`.data` The data to be processed. Can be data.frame, data.table, or a list of these objects.

Every object must have columns in the immunarch compatible format. immunarch_data_format

Competent users may provide advanced data representations: DBI database connections, Apache Spark DataFrame from copy_to or a list of these objects. They are supported with the same limitations as basic objects.

Note: each connection must represent a separate repertoire.

`.col` A string that specifies the column(s) to be processed. Outputs one of the following strings, separated by the plus sign: "nt" for nucleotide sequences, "aa" for amino acid sequences, "v" for V gene segments, "j" for J gene segments. E.g., pass "aa+v" to compute overlaps on CDR3 amino acid sequences paired with V gene segments, i.e., in this case a unique clonotype is a pair of CDR3 amino acid and V gene segment.

`.quant` A string that specifies the column to be processed. Set "count" to see public clonotype sharing with the number of clones, set "prop" to see proportions.

`.coding` Logical. If TRUE then preprocesses the data to filter out non-coding sequences.

`.min.samples` Integer. A minimal number of samples a clonotype must have to be included in the public repertoire table.

`.max.samples` Integer. A maxminal number of samples a clonotype must have to be included in the public repertoire table. Set NA (by default) to have the maximal amount of samples.

`.verbose` Logical. If TRUE then outputs the progress.

Value

Data table with columns for:

- Clonotypes (e.g., CDR3 sequence, or two columns for CDR3 sequence and V gene)

- Incidence of clonotypes

- Per-sample proportions or counts

Examples

```
# Subset the data to make the example faster to run
immdata$data <- lapply(immdata$data, head, 2000)
pr <- pubRep(immdata$data, .verbose = FALSE)
vis(pr, "clonotypes", 1, 2)
```

`pubRepApply` *Apply transformations to public repertoires*

Description

Work In Progress

Usage

```
pubRepApply(.pr1, .pr2, .fun = function(x) log10(x[1])/log10(x[2]))
```

Arguments

`.pr1` First public repertoire.

`.pr2` Second public repertoire.

`.fun` A function to apply to pairs of frequencies of same clonotypes from "pr1" and "pr2". By default - `log(X) / log(Y)` where `X,Y` - frequencies of the same clonotype, found in both public repertoires.

Value

Work in progress.

Examples

```
data(immdata)
immdata$data <- lapply(immdata$data, head, 2000)
pr <- pubRep(immdata$data, .verbose = FALSE)
pr1 <- pubRepFilter(pr, immdata$meta, .by = c(Status = "MS"))
pr2 <- pubRepFilter(pr, immdata$meta, .by = c(Status = "C"))
prapp <- pubRepApply(pr1, pr2)
head(prapp)
```

pubRepFilter *Filter out clonotypes from public repertoires*

Description

Filter our clonotypes with low incidence in a specific group.

Usage

```
pubRepFilter(.pr, .meta, .by, .min.samples = 1)
```

Arguments

`.pr`	Public repertoires, an output from pubRep.
`.meta`	Metadata file.
`.by`	Named character vector. Names of the group to filter by.
`.min.samples`	Integer. Filters out clonotypes with the number of samples below than this number.

Value

Data frame with filtered clonotypes.

Examples

```
data(immdata)
immdata$data <- lapply(immdata$data, head, 2000)
pr <- pubRep(immdata$data, .verbose = FALSE)
pr1 <- pubRepFilter(pr, immdata$meta, .by = c(Status = "MS"))
head(pr1)
```

pubRepStatistics *Statistics of number of public clonotypes for each possible combinations of repertoires*

Description

Statistics of number of public clonotypes for each possible combinations of repertoires

Usage

```
pubRepStatistics(.data, .by = NA, .meta = NA)
```

Arguments

`.data` Public repertoire, an output from the pubRep function.

`.by` Work in Progress.

`.meta` Work in Progress.

Value

Data frame with incidence statistics per sample.

Examples

```
data(immdata)
immdata$data <- lapply(immdata$data, head, 2000)
pr <- pubRep(immdata$data, .verbose = FALSE)
pubRepStatistics(pr) %>% vis()
```

`repAlignLineage` *Aligns all sequences incliding germline within each clonal lineage within each cluster*

Description

This function aligns all sequences (incliding germline) that belong to one clonal lineage and one cluster. After clustering and building the clonal lineage and germline, the next step is to analyze the degree of mutation and maturity of each clonal lineage. This allows for finding high mature cells and cells with a large number of offspring. The phylogenetic analysis will find mutations that increase the affinity of BCR. Making alignment of the sequence is the first step towards sequence analysis including BCR.

Usage

```
repAlignLineage(.data, .min_lineage_sequences, .prepare_threads, .align_threads, .nofail)
```

Arguments

`.data` The data to be processed. Can be data.frame, data.table or a list of these objects.

`.min_lineage_sequences`
If number of sequences in the same clonal lineage and the same cluster (not including germline) is lower than this threshold, this group of sequences will be filtered out from the dataframe; so only large enough lineages will be included.

`.prepare_threads`
Number of threads to prepare results table. Please note that high number can cause heavy memory usage!

`.align_threads` Number of threads for lineage alignment.

It must have columns in the immunarch compatible format immunarch_data_format, and also must contain 'Cluster' column, which is added by seqCluster() function, and 'Germline.sequence' column, which is added by repGermline() function.

`.nofail` Will return NA instead of stopping if Clustal W is not installed. Used to avoid raising errors in examples on computers where Clustal W is not installed.

Value

Dataframe or list of dataframes (if input is a list with multiple samples). The dataframe has these columns: * Cluster: cluster name * Germline: germline sequence * Alignment: DNAbin object with alignment * Sequences: nested dataframe containing all sequences for this combination of cluster and germline; it has columns * Sequence, CDR1.nt, CDR2.nt, CDR3.nt, FR1.nt, FR2.nt, FR3.nt, FR4.nt, V.allele, J.allele, V.aa, J.aa: all values taken from the input dataframe * Clone.ID: taken from the input dataframe, or created (filled with row numbers) if missing * Clones: taken from the input dataframe, or created (filled with '1' values) if missing

Examples

```
data(bcrdata)
bcr_data <- bcrdata$data

bcr_data %>%
  seqCluster(seqDist(bcr_data), .fixed_threshold = 3) %>%
  repGermline(.threads = 1) %>%
  repAlignLineage(.min_lineage_sequences = 2, .align_threads = 2, .nofail = TRUE)
```

`repClonalFamily` *Builds a phylogenetic tree using the sequences of a clonal lineage*

Description

This function uses the PHYLIP package to make phylogenetic analysis. For making trees it uses maximum parsimony methods.

Usage

```
repClonalFamily(.data, .vis_groups, .threads, .nofail)
```

Arguments

`.data` The data to be processed, output of repAlignLineage() function.

`.vis_groups` Groups for visualization, used to annotate specific clones on chart and display them in different colors. This is a named list, where names are for the chart legend, and list items are clone IDs that belong to the groups. It's not necessary to assign groups to all clonotypes; unassigned ones will be displayed on the chart as "Clonotype" category. It's also possible to assign multiple clonotypes to the same group by providing nested lists or vectors of clone IDs instead of single clone IDs. Example: .vis_groups = list(A = 817, B = 201, C = list(303, 42))

`.threads` Number of threads to use.

`.nofail` Returns NA instead of stopping if PHYLIP is not installed. Used to avoid raising errors in examples on computers where PHYLIP is not installed.

Value

Dataframe or list of dataframes (if input is a list with multiple samples). The dataframe has these columns: * Cluster: cluster name * Germline.Input: germline sequence, like it was in the input; not aligned * Germline.Output: germline sequence, parsed from PHYLIP dnapars function output; it contains difference of germline from the common ancestor; "." characters mean matching letters * Common.Ancestor: common ancestor sequence, parsed from PHYLIP dnapars function output * Trunk.Length: mean trunk length, representing the distance between the most recent common ancestor and germline sequence as a measure of the maturity of a lineage * Tree: output tree in "phylo" format, loaded from by PHYLIP dnapars function output * TreeStats: nested dataframe containing data about tree nodes, needed for visualization * Sequences: nested dataframe containing all sequences for this combination of cluster and germline; it contains regions from original sequences, saved for repSomaticHypermutation() calculation, and also data needed for visualizations

Examples

```
data(bcrdata)
bcr_data <- bcrdata$data

bcr_data %>%
  seqCluster(seqDist(bcr_data), .fixed_threshold = 3) %>%
  repGermline(.threads = 1) %>%
  repAlignLineage(.min_lineage_sequences = 2, .align_threads = 2, .nofail = TRUE) %>%
  repClonalFamily(.threads = 1, .nofail = TRUE)
```

repClonality *Clonality analysis of immune repertoires*

Description

`repClonality` function encompasses several methods to measure clonal proportions in a given repertoire.

Usage

```
repClonality(
  .data,
  .method = c("clonal.prop", "homeo", "top", "rare"),
  .perc = 10,
  .clone.types = c(Rare = 1e-05, Small = 1e-04, Medium = 0.001, Large = 0.01,
    Hyperexpanded = 1),
  .head = c(10, 100, 1000, 3000, 10000, 30000, 1e+05),
  .bound = c(1, 3, 10, 30, 100)
)
```

Arguments

`.data` The data to be processed. Can be data.frame, data.table, or a list of these objects.

Every object must have columns in the immunarch compatible format. immunarch_data_format

Competent users may provide advanced data representations: DBI database connections, Apache Spark DataFrame from copy_to or a list of these objects. They are supported with the same limitations as basic objects.

Note: each connection must represent a separate repertoire.

`.method` A String with one of the following options: `"clonal.prop"`, `"homeo"`, `"top"` or `"rare"`.

Set `"clonal.prop"` to compute clonal proportions or in other words percentage of clonotypes required to occupy specified by `.perc` percent of the total immune repertoire.

Set `"homeo"` to analyse relative abundance (also known as clonal space homeostasis), which is defined as the proportion of repertoire occupied by clonal groups with specific abundances..

Set `"top"` to estimate relative abundance for the groups of top clonotypes in repertoire, e.g., ten most abundant clonotypes. Use `".head"` to define index intervals, such as 10, 100 and so on.

Set `"rare"` to estimate relative abundance for the groups of rare clonotypes with low counts. Use `".bound"` to define the threshold of clonotype groups.

`.perc` A single numerical value ranging from 0 to 100.

`.clone.types` A named numerical vector with the threshold of the half-closed intervals that mark off clonal groups.

`.head` A numerical vector with ranges of the top clonotypes.

`.bound` A numerical vector with ranges of abundance for the rare clonotypes in the dataset.

Details

Clonal proportion assessment is a different approach to estimate repertoire diversity. When visualised, it allows for thorough examination of immune repertoire structure and composition.

In its core this type of analysis is similar to the relative species abundance concept in ecology. Relative abundance is the percent composition of an organism of a particular kind relative to the total number of organisms in the area.

A stacked barplot of relative clonotype abundances can be therefore viewed as a non-parametric approach to comparing their underlying distributions.

Value

If input data is a single immune repertoire, then the function returns a numeric vector with clonality statistics.

Otherwise, it returns a numeric matrix with clonality statistics for all input repertoires.

See Also

repDiversity

Examples

```
# Load the data
data(immdata)

imm_pr <- repClonality(immdata$data, .method = "clonal.prop")
vis(imm_pr)

imm_top <- repClonality(immdata$data, .method = "top", .head = c(10, 100, 1000, 3000, 10000))
vis(imm_top)

imm_rare <- repClonality(immdata$data, .method = "rare")
vis(imm_rare)

imm_hom <- repClonality(immdata$data, .method = "homeo")
vis(imm_hom)
```

`repDiversity` *The main function for immune repertoire diversity estimation*

Description

This is a utility function to estimate the diversity of species or objects in the given distribution.

Note: functions will check if .data is a distribution of a random variable (sum == 1) or not. To force normalisation and / or to prevent this, set .do.norm to TRUE (do normalisation) or FALSE (don't do normalisation), respectively.

Usage

```
repDiversity(
  .data,
  .method = "chao1",
  .col = "aa",
  .max.q = 6,
  .min.q = 1,
  .q = 5,
  .step = NA,
  .quantile = c(0.025, 0.975),
  .extrapolation = NA,
  .perc = 50,
  .norm = TRUE,
  .verbose = TRUE,
  .do.norm = NA,
  .laplace = 0
)
```

Arguments

`.data` The data to be processed. Can be data.frame, data.table, or a list of these objects.

Every object must have columns in the immunarch compatible format. immunarch_data_format

Competent users may provide advanced data representations: DBI database connections, Apache Spark DataFrame from copy_to or a list of these objects. They are supported with the same limitations as basic objects.

Note: each connection must represent a separate repertoire.

`.method` Picks a method used for estimation out of a following list: chao1, hill, div, gini.simp, inv.simp, gini, raref, d50, dxx.

`.col` A string that specifies the column(s) to be processed. Pass one of the following strings, separated by the plus sign: "nt" for nucleotide sequences, "aa" for amino acid sequences, "v" for V gene segments, "j" for J gene segments. E.g., pass "aa+v" to compute diversity estimations on CDR3 amino acid sequences paired with V gene segments, i.e., in this case a unique clonotype is a pair of CDR3 amino acid and V gene segment. Clonal counts of equal clonotypes will be summed up.

`.max.q` The max hill number to calculate (default: 5).

`.min.q` Function calculates several hill numbers. Set the min (default: 1).

`.q` q-parameter for the Diversity index.

`.step` Rarefaction step’s size.

`.quantile` Numeric vector with quantiles for confidence intervals.

`.extrapolation` An integer. An upper limit for the number of clones to extrapolate to. Pass 0 (zero) to turn extrapolation subroutines off.

`.perc` Set the percent to dXX index measurement.

`.norm` Normalises rarefaction curves.

`.verbose` If TRUE then outputs progress.

`.do.norm` One of the three values - NA, TRUE or FALSE. If NA then checks for distrubution (sum(.data) == 1) and normalises if needed with the given laplace correction value. if TRUE then does normalisation and laplace correction. If FALSE then doesn't do neither normalisaton nor laplace correction.

`.laplace` A numeric value, which is used as a pseudocount for Laplace smoothing.

Details

- True diversity, or the effective number of types, refers to the number of equally-abundant types needed for the average proportional abundance of the types to equal that observed in the dataset of interest where all types may not be equally abundant.

- Inverse Simpson index is the effective number of types that is obtained when the weighted arithmetic mean is used to quantify average proportional abundance of types in the dataset of interest.

- The Gini coefficient measures the inequality among values of a frequency distribution (for example levels of income). A Gini coefficient of zero expresses perfect equality, where all values are the same (for example, where everyone has the same income). A Gini coefficient of one (or 100 percents) expresses maximal inequality among values (for example where only one person has all the income).

- The Gini-Simpson index is the probability of interspecific encounter, i.e., probability that two entities represent different types.

- Chao1 estimator is a nonparameteric asymptotic estimator of species richness (number of species in a population).

- Rarefaction is a technique to assess species richness from the results of sampling through extrapolation.

- Hill numbers are a mathematically unified family of diversity indices (differing among themselves only by an exponent q).

- d50 is a recently developed immune diversity estimate. It calculates the minimum number of distinct clonotypes amounting to greater than or equal to 50 percent of a total of sequencing reads obtained following amplification and sequencing

- dXX is a similar to d50 index where XX corresponds to desirable percent of total sequencing reads.

Value

div, gini, gini.simp, inv.simp, raref return numeric vector of length 1 with value.

chao1 returns 4 values: estimated number of species, standart deviation of this number and two 95

hill returns a vector of specified length `.max.q - .min.q`

For most methods, if input data is a single immune repertoire, then the function returns a numeric vector with diversity statistics.

Otherwise, it returns a numeric matrix with diversity statistics for all input repertoires.

For Chao1 the function returns a matrix with diversity estimations.

For rarefaction the function returns either a matrix with diversity estimatinos on different step of the simulaiton process or a list with such matrices.

See Also

repOverlap, entropy, repClonality Rarefaction wiki https://en.wikipedia.org/wiki/Rarefaction_(ecology) Hill numbers paper https://www.uvm.edu/~ngotelli/manuscriptpdfs/ChaoHill.pdf Diversity wiki https://en.wikipedia.org/wiki/Measurement_of_biodiversity

Examples

```
data(immdata)

# Make data smaller for testing purposes
immdata$data <- top(immdata$data, 4000)

# chao1
repDiversity(.data = immdata$data, .method = "chao1") %>% vis()

# Hill numbers
repDiversity(
  .data = immdata$data, .method = "hill", .max.q = 6,
  .min.q = 1, .do.norm = NA, .laplace = 0
) %>% vis()

# diversity
repDiversity(.data = immdata$data, .method = "div", .q = 5, .do.norm = NA, .laplace = 0) %>%
  vis()

# Gini-Simpson
repDiversity(.data = immdata$data, .method = "gini.simp", .q = 5, .do.norm = NA, .laplace = 0) %>%
  vis()

# inverse Simpson
repDiversity(.data = immdata$data, .method = "inv.simp", .do.norm = NA, .laplace = 0) %>% vis()

# Gini coefficient
repDiversity(.data = immdata$data, .method = "gini", .do.norm = NA, .laplace = 0)

# d50
repDiversity(.data = immdata$data, .method = "d50") %>% vis()
```

---

repExplore — *Main function for exploratory data analysis: compute the distribution of lengths, clones, etc.*

Description

The `repExplore` function calculates the basic statistics of repertoire: the number of unique immune receptor clonotypes, their relative abundances, and sequence length distribution across the input dataset.

Usage

```
repExplore(
  .data,
  .method = c("volume", "count", "len", "clones"),
  .col = c("nt", "aa"),
  .coding = TRUE
)
```

Arguments

`.data` The data to be processed. Can be data.frame, data.table, or a list of these objects.

Every object must have columns in the immunarch compatible format. immunarch_data_format

Competent users may provide advanced data representations: DBI database connections, Apache Spark DataFrame from copy_to or a list of these objects. They are supported with the same limitations as basic objects.

Note: each connection must represent a separate repertoire.

`.method` A string that specifies the method of analysis. It can be either "volume", "count", "len" or "clones".

When .method is set to "volume" the repExplore calculates the number of unique clonotypes in the input data.

When .method is set to "count" the repExplore calculates the distribution of clonotype abundances, i.e., how frequent receptors with different abundances are.

When .method is set to "len" the repExplore calculates the distribution of CDR3 sequence lengths.

When .method is set to "clones" the repExplore returns the number of clones (i.e., cells) per input repertoire.

`.col` A string that specifies the column to be processed. Pass "nt" for nucleotide sequence or "aa" for amino acid sequence.

`.coding` If `TRUE`, then only coding sequences will be analysed.

Value

If input data is a single immune repertoire, then the function returns a numeric vector with exploratory analysis statistics.

Otherwise, it returns a numeric matrix with exploratory analysis statistics for all input repertoires.

See Also

vis.immunr_exp_vol

Examples

```
data(immdata)

# Calculate statistics and generate a visual output with vis()
```

```
repExplore(immdata$data, .method = "volume") %>% vis()

repExplore(immdata$data, .method = "count") %>% vis()

repExplore(immdata$data, .method = "len") %>% vis()
```

`repFilter` *Main function for data filtering*

Description

Main function for data filtering

Usage

```
repFilter(
  .data,
  .method = "by.clonotype",
  .query = list(CDR3.aa = exclude("partial", "out_of_frame")),
  .match = "exact"
)
```

Arguments

`.data` The data to be processed. Must be the list of 2 elements: a data table and a metadata table.

`.method` Method of filtering. Implemented methods: by.meta, by.repertoire (by.rep), by.clonotype (by.cl) Default value: 'by.clonotype'.

`.query` Filtering query. It's a named list of filters that will be applied to data. Possible values for names in this list are dependent on filter methods: - by.meta: filters by metadata. Names in the named list are metadata column headers. - by.repertoire: filters by the number of clonotypes or total number of clones in sample. Possible names in the named list are "n_clonotypes" and "n_clones". - by.clonotype: filters by data in all samples. Names in the named list are data column headers. Elements of the named list for each of the filters are filtering options. Possible values for filtering options: - include("STR1", "STR2", ...): keeps only rows with matching values. Available for methods: "by.meta", "by.clonotype". - exclude("STR1", "STR2", ...): removes rows with matching values. Available for methods: "by.meta", "by.clonotype". - lessthan(value): keeps rows/samples with numeric values less than specified. Available for methods: "by.meta", "by.repertoire", "by.clonotype". - morethan(value): keeps rows/samples with numeric values more than specified. Available for methods: "by.meta", "by.repertoire", "by.clonotype". - interval(from, to): keeps rows/samples with numeric values that fits in this interval. from is inclusive, to is exclusive. Available for methods: "by.meta", "by.repertoire", "by.clonotype". Default value: 'list(CDR3.aa = exclude("partial", "out_of_frame"))'.

`.match` Matching method for "include" and "exclude" options in query. Possible values: - exact: matches only the exact specified string; - startswith: matches all strings starting with the specified substring; - substring: matches all strings containing the specified substring. Default value: 'exact'.

Examples

```
data(immdata)

# Select samples with status "MS"
repFilter(immdata, "by.meta", list(Status = include("MS")))

# Select samples without status "MS"
repFilter(immdata, "by.meta", list(Status = exclude("MS")))

# Select samples from lanes "A" and "B" with age > 15
repFilter(immdata, "by.meta", list(Lane = include("A", "B"), Age = morethan(15)))

# Select samples that are not from lanes "A" and "B"
repFilter(immdata, "by.meta", list(Lane = exclude("A", "B")))

# Select samples with a number of clonotypes from 1000 to 5000
repFilter(immdata, "by.repertoire", list(n_clonotypes = interval(1000, 5000)))

# Select clonotypes in all samples with alpha chains
repFilter(immdata, "by.clonotype",
  list(V.name = include("AV"), J.name = include("AJ")),
  .match = "substring"
)
```

`repGermline` *Creates germlines for clonal lineages*

Description

This function creates germlines for clonal lineages. B cell clonal lineage represents a set of B cells that presumably have a common origin (arising from the same VDJ rearrangement event) and a common ancestor. Each clonal lineage has its own germline sequence that represents the ancestral sequence for each BCR in clonal lineage. In other words, germline sequence is a sequence of B-cells immediately after VDJ recombination, before B-cell maturation and hypermutation process. Germline sequence is useful for assessing the degree of mutation and maturity of the repertoire.

Usage

```
repGermline(.data, .species, .min_nuc_outside_cdr3, .threads)
```

Arguments

.data The data to be processed. Can be data.frame, data.table or a list of these objects.

It must have columns in the immunarch compatible format immunarch_data_format.

.species Species from which the data was acquired. Available options: "HomoSapiens" (default), "MusMusculus", "BosTaurus", "CamelusDromedarius", "CanisLupusFamiliaris", "DanioRerio", "MacacaMulatta", "MusMusculusDomesticus", "MusMusculusCastaneus", "MusMusculusMolossinus", "MusMusculusMusculus", "MusSpretus", "OncorhynchusMykiss", "OrnithorhynchusAnatinus", "OryctolagusCuniculus", "RattusNorvegicus", "SusScrofa".

.min_nuc_outside_cdr3

This parameter sets how many nucleotides should have V or J chain outside of CDR3 to be considered good for further alignment.

.threads Number of threads to use.

Value

Data with added columns: * Sequence (FR1+CDR1+FR2+CDR2+FR3+CDR3+FR4 in nucleotides; the column will be replaced if exists) * V.allele, J.allele (chosen alleles of V and J genes), * V.aa, J.aa (V and J sequences from original clonotype, outside CDR3, converted to amino acids) * Germline.sequence (combined germline nucleotide sequence)

Examples

```
data(bcrdata)

bcrdata$data %>%
  top(5) %>%
  repGermline()
```

---

repLoad *Load immune repertoire files into the R workspace*

Description

The repLoad function loads repertoire files into R workspace in the immunarch format where you can immediately use them for the analysis. repLoad automatically detects the right format for your files, so all you need is simply provide the path to your files.

See "Details" for more information on supported formats. See "Examples" for diving right into it.

Usage

```
repLoad(.path, .mode = "paired", .coding = TRUE, ...)
```

Arguments

`.path` A character string specifying the path to the input data. Input data can be one of the following:

- a single repertoire file. In this case `repLoad` returns an R data.frame;

- a vector of paths to repertoire files. Same as in the case with no metadata file presented in the next section below;

- a path to the folder with repertoire files and, if available, metadata file "metadata.txt". If the metadata file if presented, then the `repLoad` returns a list with two elements "data" and "meta". "data" is an another list with repertoire R data.frames. "meta" is a data frame with the metadata. If the metadata file "metadata.txt" is not presented, then the `repLoad` creates a dummy metadata file with sample names and returns a list with two elements "data" and "meta". If input data has multiple chains or cell types stored in the same file (for example, like in 10xGenomics repertoire files), such repertoire files will be splitted to different R data frames with only one type of chain and cell presented. The metadata file will have additional columns specifying cell and chain types for different samples.

`.mode` Either "single" for single chain data or "paired" for paired chain data.

Currently "single" works for every format, and "paired" works only for 10X Genomics data.

By default, 10X Genomics data will be loaded as paired chain data, and other files will be loaded as single chain data.

`.coding` A logical value. Set TRUE to get coding-only clonotypes (by defaul). Set FALSE to get all clonotypes.

`...` Extra arguments for parsing functions

Details

The metadata has to be a tab delimited file with first column named "Sample". It can have any number of additional columns with arbitrary names. The first column should contain base names of files without extensions in your folder. Example:

Sample	Sex	Age	Status
immunoseq_1	M	1	C
immunoseq_2	M	2	C
immunoseq_3	FALSE	3	A

Currently, Immunarch support the following formats:

- "immunoseq" - ImmunoSEQ of any version. http://www.adaptivebiotech.com/immunoseq

- "mitcr" - MiTCR. https://github.com/milaboratory/mitcr

- "mixcr" - MiXCR (the "all" files) of any version. https://github.com/milaboratory/mixcr

- "migec" - MiGEC. http://migec.readthedocs.io/en/latest/

- "migmap" - For parsing IgBLAST results postprocessed with MigMap. https://github.com/mikessh/migmap

- "tcr" - tcR, our previous package. https://imminfo.github.io/tcr/

- "vdjtools" - VDJtools of any version. http://vdjtools-doc.readthedocs.io/en/latest/
- "imgt" - IMGT HighV-QUEST. http://www.imgt.org/HighV-QUEST/
- "airr" - adaptive immune receptor repertoire (AIRR) data format. http://docs.airr-community.org/en/latest/datarep/overview.
- "10x" - 10XGenomics clonotype annotations tables. https://support.10xgenomics.com/single-cell-vdj/software/pipelines/latest/output/annotation
- "archer" - ArcherDX clonotype tables. https://archerdx.com/

Value

A list with two named elements:

- "data" is a list of input samples;
- "meta" is a data frame with sample metadata.

See Also

immunr_data_format for immunarch data format; repSave for file saving; repOverlap, geneUsage and repDiversity for starting with immune repertoires basic statistics.

Examples

```
# To load the data from a single file (note that you don't need to specify the data format):
file_path <- paste0(system.file(package = "immunarch"), "/extdata/io/Sample1.tsv.gz")
immdata <- repLoad(file_path)

# Suppose you have a following structure in your folder:
# >_ ls
# immunoseq1.txt
# immunoseq2.txt
# immunoseq3.txt
# metadata.txt

# To load the whole folder with every file in it type:
file_path <- paste0(system.file(package = "immunarch"), "/extdata/io/")
immdata <- repLoad(file_path)
print(names(immdata))

# We recommend creating a metadata file named "metadata.txt" in the folder.

# In that case, when you load your data you will see:
# > immdata <- repLoad("path/to/your/folder/")
# > names(immdata)
# [1] "data" "meta"

# If you do not have "metadata.txt", you will see the same output,
# but your metadata will be almost empty:
# > immdata <- repLoad("path/to/your/folder/")
# > names(immdata)
# [1] "data" "meta"
```

repOverlap *Main function for public clonotype statistics calculations*

Description

The `repOverlap` function is designed to analyse the overlap between two or more repertoires. It contains a number of methods to compare immune receptor sequences that are shared between individuals.

Usage

```
repOverlap(
  .data,
  .method = c("public", "overlap", "jaccard", "tversky", "cosine", "morisita",
    "inc+public", "inc+morisita"),
  .col = "aa",
  .a = 0.5,
  .b = 0.5,
  .verbose = TRUE,
  .step = 1000,
  .n.steps = 10,
  .downsample = FALSE,
  .bootstrap = NA,
  .verbose.inc = NA,
  .force.matrix = FALSE
)
```

Arguments

`.data` The data to be processed. Can be data.frame, data.table, or a list of these objects.

Every object must have columns in the immunarch compatible format. immunarch_data_format

Competent users may provide advanced data representations: DBI database connections, Apache Spark DataFrame from copy_to or a list of these objects. They are supported with the same limitations as basic objects.

Note: each connection must represent a separate repertoire.

`.method` A string that specifies the method of analysis or a combination of methods. The `repOverlap` function supports following basic methods: "public", "overlap", "jaccard", "tversky", "cosine", "morisita". If vector of multiple methods is given for this parameter, the first method will be used.

`.col` A string that specifies the column(s) to be processed. Pass one of the following strings, separated by the plus sign: "nt" for nucleotide sequences, "aa" for amino acid sequences, "v" for V gene segments, "j" for J gene segments. E.g., pass "aa+v" to compute overlaps on CDR3 amino acid sequences paired with V gene segments, i.e., in this case a unique clonotype is a pair of CDR3 amino acid and V gene segment. Clonal counts of equal clonotypes will be summed up.

`.a, .b` Alpha and beta parameters for Tversky Index. Default values give the Jaccard index measure.

`.verbose` if TRUE then output the progress.

`.step` Either an integer or a numeric vector.

In the first case, the integer defines the step of incremental overlap.

In the second case, the vector encodes all repertoire sampling depths.

`.n.steps` Skipped if ".step" is a numeric vector.

`.downsample` If TRUE then performs downsampling to N clonotypes at each step instead of choosing the top N clonotypes in incremental overlaps. Change nothing of you are using conventional methods.

`.bootstrap` Set NA to turn off any bootstrapping, set a number to perform bootstrapping with this number of tries.

`.verbose.inc` Logical. If TRUE then shows output from the computation process.

`.force.matrix` Logical. If TRUE then always forces the matrix output even in case of two input repertoires.

Details

"public" and "shared" are synonyms that exist for the convenience of researchers.

The "overlap" coefficient is a similarity measure that measures the overlap between two finite sets.

The "jaccard" index is conceptually a percentage of how many objects two sets have in common out of how many objects they have total.

The "tversky" index is an asymmetric similarity measure on sets that compares a variant to a prototype.

The "cosine" index is a measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them.

The "morisita" index measures how many times it is more likely to randomly select two sampled points from the same quadrat (the dataset is covered by a regular grid of changing size) then it would be in the case of a random distribution generated from a Poisson process. Duplicate objects are merged with their counts are summed up.

Value

In most cases the return value is a matrix with overlap values for each pair of repertoires.

If only two repertoires were provided, return value is single numeric value.

If one of the incremental method is chosen, return list of overlap matrix.

See Also

inc_overlap, vis

Examples

```
data(immdata)

# Make data smaller for testing purposes
immdata$data <- top(immdata$data, 4000)

ov <- repOverlap(immdata$data, .verbose = FALSE)
vis(ov)

ov <- repOverlap(immdata$data, "jaccard", .verbose = FALSE)
vis(ov, "heatmap2")
```

repOverlapAnalysis *Post-analysis of public clonotype statistics: PCA, clustering, etc.*

Description

The repOverlapAnalysis function contains advanced data analysis methods. You can use several clustering and dimensionality reduction techniques in order to investigate further the difference between repertoires provided.

To cluster a subset of similar data with repOverlapAnalysis you can perform hierarchical clustering, k-means or dbscan ('hclust', 'kmeans', 'dbscan' respectively).

To reduce dimensions, for example, to select features for subsequent analysis, you can execute the multidimensional scaling or t-sne algorithms ('mds' and 'tsne' respectively).

Usage

```
repOverlapAnalysis(
  .data,
  .method = ("hclust"),
  .scale = default_scale_fun,
  .raw = TRUE,
  .perp = 1,
  .theta = 0.1,
  .eps = 0.01,
  .k = 2
)
```

Arguments

`.data` Any distance matrix between pairs of repertoires. You can also pass your output from repOverlap.

`.method` A string that defines the type of analysis to perform.

`.scale` A function to scale the data before passing it to the MDS algorithm.

`.raw` A logical value. Set TRUE if you want to receive raw output of clustering or dimensionality reduction function of choice. Set FALSE if you want to receive processed output that can be subjected to visualisation with vis function.

`.perp` A numerical value, t-SNE parameter, see `immunr_tsne`.

`.theta` A numerical value, t-SNE parameter, see `immunr_tsne`.

`.eps` A numerical value, DBscan epsylon parameter, see `immunr_dbscan`.

`.k` The number of clusters to create, passed as `k` to hcut or as `centers` to kmeans.

Value

Depends on the last element in the `.method` string. See immunr_tsne for more info.

Examples

```
data(immdata)
ov <- repOverlap(immdata$data)
repOverlapAnalysis(ov, "mds+hclust") %>% vis()
```

---

repSample *Downsampling and resampling of immune repertoires*

Description

Sample (downsample) repertoires using different approches.

Usage

```
repSample(
  .data,
  .method = c("downsample", "resample", "sample"),
  .n = NA,
  .prob = TRUE
)
```

Arguments

`.data` The data to be processed. Can be data.frame, data.table, or a list of these objects.

Every object must have columns in the immunarch compatible format. immunarch_data_format

Competent users may provide advanced data representations: DBI database connections, Apache Spark DataFrame from copy_to or a list of these objects. They are supported with the same limitations as basic objects.

Note: each connection must represent a separate repertoire.

`.method` Character. Name of a sampling method. See "Description" for more details. Default value is "downsample" that downsamples the repertoires to the number of clones (i.e., reads / UMIs) that the smallest repertoire has, if user doesn't set any value to the ".n" argument.

`.n` Integer. Number of clones / clonotypes / reads / UMIs to choose, depending on the method. Set NA to sample repertoires to the size of the smallest repertoire in the ".data".

`.prob` Logical. If TRUE then samples the clonotypes with probability weights equal to their number of clones. Used only if ".method" is "sample".

Details

If `.method` is "downsample" then `repSample` chooses `.n` clones (not clonotypes!) from the input repertoires without any probabilistic simulation, but exactly computing each choosed clones. Such approach is is more consistent and biologically pleasant than an output from the function if `.method` is "resample".

If `.method` is "resample" then `repSample` uses multinomial distribution to compute the number of occurences for each cloneset. then it removes zero-number clonotypes and return the resulting data frame. Probabilities for `rmultinom` for each cloneset is a percentage of this cloneset in the "Proportion" column. It's a some sort of simulation of how clonotypes are chosen from the organisms.

if `.method` is "sample" then `repSample` chooses `.n` clonotypes (not clones!) randomly. Depending on the `.prob` argument, the function chooses clonotypes either according to their size (if `.prob` is TRUE, by default), or each clonotype has an equal chance to be choosed (if `.prob` is FALSE). Note that sampling is done without replacing.

Value

Subsampled immune repertoire or a list of subsampled immune repertoires.

See Also

rmultinom, clonal_proportion

Examples

```
data(immdata)
# Downsampling to 1000 clones (not clonotypes!)
tmp <- repSample(immdata$data[[1]], .n = 1000)
sum(tmp$Clones)

# Downsampling to 1000 clonotypes
tmp <- repSample(immdata$data[[1]], "sample", .n = 1000)
nrow(tmp)

# Downsampling to the smallest repertoire by clones (not clonotypes!)
tmp <- repSample(immdata$data[c(1, 2)])
sum(tmp[[1]]$Clones)
sum(tmp[[2]]$Clones)

# Downsampling to the smallest repertoire by clonotypes
tmp <- repSample(immdata$data[c(1, 2)], "sample")
nrow(tmp[[1]]$Clones)
nrow(tmp[[2]]$Clones)
```

repSave *Save immune repertoires to the disk*

Description

The repSave function is deigned to save your data to the disk in desirable format. Currently supports "immunarch" and "vdjtools" file formats.

Usage

```
repSave(.data, .path, .format = c("immunarch", "vdjtools"), .compress = TRUE)
```

Arguments

.data	An R dataframe, a list of R dataframes or a list with data and meta where first element is a list of dataframes and the latter is a dataframe with metadata.
.path	A string with the path to the output directory. It should include file name if a single dataframe is provided to .data argument.
.format	A string with desirable format specification. Current options are "immunarch" and "vdjtools".
.compress	A boolean value. Defines whether the output will be compressed or not.

Details

It is not necessary to create directories beforehand. If the provided directory does not exist it will be created automatically.

Value

No return value.

Examples

```
data(immdata)
# Reduce data to save time on examples
immdata$data <- purrr::map(immdata$data, ~ .x %>% head(10))
dirpath <- tempdir()
# Save the list of repertoires
repSave(immdata, dirpath)
# Load it and check if it is the same
new_immdata <- repLoad(dirpath)
# sum(immdata$data[[1]] != new_immdata$data[[1]], na.rm = TRUE)
# sum(immdata$data[[2]] != new_immdata$data[[2]], na.rm = TRUE)
# sum(immdata$meta != new_immdata$meta, na.rm = TRUE)
```

repSomaticHypermutation

Calculates number of mutations against the germline for each clonotype

Description

This function aligns V and J genes from the germline in each cluster with corresponding genes in each clonotype, saves the alignments for purpose of visualization, and calculates number of mutations for each clonotype.

Usage

```
repSomaticHypermutation(.data, .threads, .nofail)
```

Arguments

`.data`	The data to be processed: an output of repClonalFamily(); variants with one sample and list of samples are both supported.
`.threads`	Number of threads to use.
`.nofail`	Will return NA instead of stopping if Clustal W is not installed. Used to avoid raising errors in examples on computers where Clustal W is not installed.

Value

Dataframe or list of dataframes (if input is a list with multiple samples). The dataframe has all the columns from repClonalFamily() output dataframe, with Sequence column unnested: the resulting dataframe has one line per clonotype. Clone.ID column contains original IDs for clonotypes, and can be used as dataframe key. New columns are added: * Germline.Alignment.V: contains V gene alignment of current clonotype with the germline * Germline.Alignment.J: contains J gene alignment of current clonotype with the germline * Substitutions: contains number of substitutions in the alignment (summary for V and J) * Insertions: contains number of insertions in the clonotype relative to germline (summary for V and J) * Deletions: contains number of deletions in the clonotype relative to germline (summary for V and J) * Mutations: contains total number of mutations in the alignment (summary for V and J)

Examples

```
data(bcrdata)
bcr_data <- bcrdata$data

bcr_data %>%
  seqCluster(seqDist(bcr_data), .fixed_threshold = 3) %>%
  repGermline(.threads = 1) %>%
  repAlignLineage(.min_lineage_sequences = 2, .align_threads = 2, .nofail = TRUE) %>%
  repClonalFamily(.threads = 1, .nofail = TRUE) %>%
  repSomaticHypermutation(.threads = 1, .nofail = TRUE)
```

`scdata` *Paired chain immune repertoire dataset*

Description

A dataset with paired chain IG data for testing and examplatory purposes.

Usage

```
scdata
```

Format

A list of four elements: "data" is a list with data frames with clonotype tables. "meta" is a metadata table. "bc_patients" is a list of barcodes corresponding to specific patients. "bc_clusters" is a list of barcodes corresponding to specific cell clusters.

data List of immune repertoire data frames.

meta Metadata ...

`select_barcodes` *Select specific clonotypes using barcodes from single-cell metadata*

Description

Subsets the input immune repertoire by barcodes. Creates a vector of barcodes to subset or a vector cluster IDs and corresponding barcodes to get a list of immune repertoires corresponding to cluster IDs. Columns with clonotype counts and proportions are changed accordingly to the filtered barcodes.

Usage

```
select_barcodes(.data, .barcodes, .force.list = FALSE)
```

Arguments

`.data` The data to be processed. Can be data.frame, data.table, or a list of these objects.

Every object must have columns in the immunarch compatible format. immunarch_data_format

Competent users may provide advanced data representations: DBI database connections, Apache Spark DataFrame from copy_to or a list of these objects. They are supported with the same limitations as basic objects.

Note: each connection must represent a separate repertoire.

`.barcodes` Either a character vector with barcodes or a named character/factor vector with barcodes as names and cluster IDs a vector elements. The output of Seurat's `Idents` function works.

`.force.list` Logical. If TRUE then always returns a list, even if the result is one data frame.

Value

An immune repertoire (if ".barcodes" is a barcode vector) or a list of immune repertoires (if ".barcodes" is named vector or an output from Seurat::Idents()). Each element is an immune repertoire with clonotype barcodes corresponding to the input barcodes. The output list names are cluster names in the ".barcode" argument (Seurat::Idents() case only).

See Also

select_clusters

Examples

```
## Not run:
data(immdata)
# Create a fake single-cell data
df <- immdata$data[[1]]
df$Barcode <- "AAAAACCCCC"
df$Barcode[51:nrow(df)] <- "GGGGGCCCCC"
barcodes <- "AAAAACCCCC"
df <- select_barcodes(df, barcodes)
nrow(df)

## End(Not run)
```

---

select_clusters *Split the immune repertoire data to clusters from single-cell barcodes*

Description

Given the vector of barcodes from Seurat, splits the input repertoires to separate subsets following the barcodes' assigned IDs. Useful in case you want to split immune repertoires by patients or clusters.

Usage

```
select_clusters(.data, .clusters, .field = "Cluster")
```

Arguments

.data	List of two elements "data" and "meta", with "data" being a list of immune repertoires, and "meta" being a metadata table.
.clusters	Factor vector with barcodes as vector names and cluster IDs as vector elements. The output of the Seurat `Idents` function works.
.field	A string specifying the name of the field in the input metadata. New immune repertoire subsets will have cluster IDs in this field.

Value

A list with two elements "data" and "meta" with updated immune repertoire tables and metadata.

See Also

select_barcodes

Examples

```
## Not run:
library(Seurat)
Idents(pbmc_small)
new_cluster_ids <- c("A", "B", "C")
new_cluster_ids <- levels(pbmc_small)
new_cluster_ids
pbmc_small <- RenameIdents(pbmc_small, new_cluster_ids)

## End(Not run)
```

---

seqCluster *Function for assigning clusters based on sequences similarity*

Description

Graph clustering based on distances between sequences

Usage

```
seqCluster(.data, .dist, .perc_similarity, .nt_similarity, .fixed_threshold)
```

Arguments

`.data` The data which was used to caluculate .dist object. Can be data.frame, data.table, or a list of these objects.

Every object must have columns in the immunarch compatible format immunarch_data_format

`.dist` List of distance objects produced with seqDist function.

`.perc_similarity`
Numeric value between 0 and 1 specifying the maximum acceptable weight of an edge in a graph. This threshold depends on the length of sequences.

`.nt_similarity` Numeric between 0-sequence length specifying the threshold of allowing a 1 in n nucleotides mismatch in sequencies.

`.fixed_threshold`
Numeric specifying the threshold on the maximum weight of an edge in a graph.

Value

Immdata data format object. Same as .data, but with extra ’Cluster’ column with clusters assigned.

Examples

```
data(immdata)
# In this example, we will use only 2 samples with 500 clonotypes in each for time saving
input_data <- lapply(immdata$data[1:2], head, 500)
dist_result <- seqDist(input_data)
cluster_result <- seqCluster(input_data, dist_result, .fixed_threshold = 1)
```

`seqDist` *Function for computing distance for sequences*

Description

Computing sequential distances between clonotypes from two repertoires:

Usage

```
seqDist(.data, .col = 'CDR3.nt', .method = 'hamming',
 .group_by = c(”V.name”, ”J.name”), .group_by_seqLength = TRUE, .trim_genes = TRUE, ...)
```

Arguments

`.data` The data to be processed. Can be data.frame, data.table, or a list of these objects.

Every object must have columns in the immunarch compatible format immunarch_data_format

`.col` A string that specifies the column name to be processed. The default value is ’CDR3.nt’.

`.method` Character value or user-defined function.

`.group_by` Character vector of column names to group sequence by. The default value is c("V.first", "J.first"). Columns "V.first" and "J.first" containing first genes without allele suffixes are calculated automatically from "V.name" and "J.name" if absent in the data. Pass NA for no grouping options.

`.group_by_seqLength`

If TRUE - adds grouping by sequence length of .col argument

`.trim_genes` If TRUE - use only general gene values (e.g. "IGHV1-18") of .group_by columns for clustering; if FALSE - can cause very small clusters in case of high resolution genotyping

`...` Extra arguments for user-defined function.

The default value is `'hamming'` for Hamming distance which counts the number of character substitutions that turns b into a. If a and b have different number of characters the distance is Inf.

Other possible values are:

`'lv'` for Levenshtein distance which counts the number of deletions, insertions and substitutions necessary to turn b into a.

'lcs' for longest common substring is defined as the longest string can be obtained by pairing characters from a and b while keeping the order of characters intact.

In case of user-defined function, it should take x and y parameters as input and return dist object.

Value

Named list of list with dist objects for given repertoires for each combination of .group_by variable(s) and/or sequence length of .col.

Examples

```
data(immdata)
# Reducing data to save time on examples
immdata$data <- purrr::map(immdata$data, ~ .x %>% head(10))
# Computing hamming distance for the first two repertoires in \code{'immdata'}
seqDist(immdata$data[1:2])

# Here we define a custom distance function
# that will count the difference in number of characters in sequences.

f <- function(x, y) {
  res <- matrix(nrow = length(x), ncol = length(y))
  for (i in 1:length(x)) {
    res[i, ] <- abs(nchar(x[i]) - nchar(y))
  }
  dimnames(res) <- list(x, y)
  return(as.dist(res))
}

seqDist(immdata$data[1:2], .method = f, .group_by_seqLength = FALSE)
```

---

set_pb   *Set and update progress bars*

Description

Set and update progress bars

Usage

```
set_pb(.max)

add_pb(.pb, .value = 1)
```

Arguments

`.max`	Integer. Maximal value of the progress bar.
`.pb`	Progress bar object from `set_pb`.
`.value`	Numeric. Value to add to the progress bar at each step.

Value

An updated progress bar.

Developer Examples

pb <- immunarch:::set_pb(100) immunarch:::add_pb(pb, 25) immunarch:::add_pb(pb, 25) immunarch:::add_pb(pb, 25) immunarch:::add_pb(pb, 25) close(pb)

`spectratype` *Immune repertoire spectratyping*

Description

Immune repertoire spectratyping

Usage

```
spectratype(.data, .quant = c("id", "count"), .col = "nt")
```

Arguments

`.data`	The data to be processed. Can be data.frame, data.table, or a list of these objects. Every object must have columns in the immunarch compatible format. immunarch_data_format Competent users may provide advanced data representations: DBI database connections, Apache Spark DataFrame from copy_to or a list of these objects. They are supported with the same limitations as basic objects. Note: each connection must represent a separate repertoire.
`.quant`	Select the column with clonal counts to evaluate. Set to "id" to count every clonotype once. Set to "count" to take into the account number of clones per clonotype.
`.col`	A string that specifies the column(s) to be processed. The output is one of the following strings, separated by the plus sign: "nt" for nucleotide sequences, "aa" for amino acid sequences, "v" for V gene segments, "j" for J gene segments. E.g., pass "aa+v" for spectratyping on CDR3 amino acid sequences paired with V gene segments, i.e., in this case a unique clonotype is a pair of CDR3 amino acid and V gene segment. Clonal counts of equal clonotypes will be summed up.

Value

Data frame with distributions of clonotypes per CDR3 length.

Examples

```
# Load the data
data(immdata)
sp <- spectratype(immdata$data[[1]], .col = "aa+v")
vis(sp)
```

`split_to_kmers` *Analysis immune repertoire kmer statistics: sequence profiles, etc.*

Description

Analysis immune repertoire kmer statistics: sequence profiles, etc.

Usage

```
split_to_kmers(.data, .k)

kmer_profile(.data, .method = c("freq", "prob", "wei", "self"), .remove.stop = TRUE)
```

Arguments

`.data`	Character vector or the output from `getKmers`.
`.k`	Integer. Size of k-mers.
`.method`	Character vector of length one. If "freq" then returns a position frequency matrix (PFM) - a matrix with occurences of each amino acid in each position. If "prob" then returns a position probability matrix (PPM) - a matrix with probabilities of occurences of each amino acid in each position. This is a traditional representation of sequence motifs. If "wei" then returns a position weight matrix (PWM) - a matrix with log likelihoods of PPM elements. If "self" then returns a matrix with self-information of elements in PWM. For more information see https://en.wikipedia.org/wiki/Position_weight_matrix.
`.remove.stop`	Logical. If TRUE (by default) remove stop codons.

Value

`split_to_kmers` - Data frame with two columns (k-mers and their counts).

`kmer_profile` - a matrix with per-position amino acid statistics.

Examples

```
data(immdata)
kmers <- getKmers(immdata$data[[1]], 5)
kmer_profile(kmers) %>% vis()
```

switch_type *Return a column's name*

Description

Return a column's name

Usage

```
switch_type(type)

process_col_argument(.col)
```

Arguments

type	Character. Specifies the column to choose: "nt" chooses the CDR3 nucleotide column, "aa" chooses the CDR3 amino acid column, "v" chooses the V gene segment column, "j" chooses the J gene segment column.
.col	A string that specifies the column(s) to be processed. Select one of the following strings, separated by the plus sign: "nt" for nucleotide sequences, "aa" for amino acid sequences, "v" for V gene segments, "j" for J gene segments.

Value

A column's name.

Developer Examples

immunarch:::switch_type("nuc") immunarch:::switch_type("v")

top *Get the N most abundant clonotypes*

Description

Get the N most abundant clonotypes

Usage

```
top(.data, .n = 10)
```

Arguments

`.data`	The data to be processed. Can be data.frame, data.table, or a list of these objects. Every object must have columns in the immunarch compatible format. immunarch_data_format Competent users may provide advanced data representations: DBI database connections, Apache Spark DataFrame from copy_to or a list of these objects. They are supported with the same limitations as basic objects. Note: each connection must represent a separate repertoire.
`.n`	Numeric. Number of the most abundant clonotypes to return.

Value

Data frame with the `.n` most abundant clonotypes only.

Examples

```
data(immdata)
top(immdata$data)
top(immdata$data[[1]])
```

`trackClonotypes` *Track clonotypes across time and data points*

Description

Tracks the temporal dynamics of clonotypes in repertoires. For example, tracking across multiple time points after vaccination.

Note: duplicated clonotypes are merged and their counts are summed up.

Usage

```
trackClonotypes(.data, .which = list(1, 15), .col = "aa", .norm = TRUE)
```

Arguments

`.data`	The data to process. It can be a data.frame, a data.table, or a list of these objects. Every object must have columns in the immunarch compatible format. immunarch_data_format Competent users may provide advanced data representations: DBI database connections, Apache Spark DataFrame from copy_to or a list of these objects. They are supported with the same limitations as basic objects. Note: each connection must represent a separate repertoire.

`.which` An argument that regulates which clonotypes to choose for tracking. There are three options for this argument:

1) passes a list with two elements `list(X, Y)`, where `X` is the name or the index of a target repertoire from ".data", and `Y` is the number of the most abundant clonotypes to take from `X`.

2) passes a character vector of sequences to take from all data frames;

3) passes a data frame (data table, database) with one or more columns - first for sequences, and other for gene segments (if applicable).

See the "Examples" below with examples for each option.

`.col` A character vector of length 1. Specifies an identifier for a column, from which the function chooses clonotype sequences. Specify "nt" for nucleotide sequences, "aa" for amino acid sequences, "aa+v" for amino acid sequences and Variable genes, "nt+j" for nucleotide sequences with Joining genes, or any combination of the above. Used only if ".which" has option 1) or option 2).

`.norm` Logical. If TRUE then uses Proportion instead of the number of Clones per clonotype to store in the function output.

Value

Data frame with input sequences and counts or proportions for each of the input repertoire.

Examples

```
# Load an example data that comes with immunarch
data(immdata)

# Make the data smaller in order to speed up the examples
immdata$data <- immdata$data[c(1, 2, 3, 7, 8, 9)]
immdata$meta <- immdata$meta[c(1, 2, 3, 7, 8, 9), ]

# Option 1
# Choose the first 10 amino acid clonotype sequences
# from the first repertoire to track
tc <- trackClonotypes(immdata$data, list(1, 10), .col = "aa")
# Choose the first 20 nucleotide clonotype sequences
# and their V genes from the "MS1" repertoire to track
tc <- trackClonotypes(immdata$data, list("MS1", 20), .col = "nt+v")

# Option 2
# Choose clonotypes with amino acid sequences "CASRGLITDTQYF" or "CSASRGSPNEQYF"
tc <- trackClonotypes(immdata$data, c("CASRGLITDTQYF", "CSASRGSPNEQYF"), .col = "aa")

# Option 3
# Choose the first 10 clonotypes from the first repertoire
# with amino acid sequences and V segments
target <- immdata$data[[1]] %>%
  select(CDR3.aa, V.name) %>%
  head(10)
tc <- trackClonotypes(immdata$data, target)
```

```
# Visualise the output regardless of the chosen option
# Therea are three way to visualise it, regulated by the .plot argument
vis(tc, .plot = "smooth")
vis(tc, .plot = "area")
vis(tc, .plot = "line")

# Visualising timepoints
# First, we create an additional column in the metadata with randomly choosen timepoints:
immdata$meta$Timepoint <- sample(1:length(immdata$data))
immdata$meta
# Next, we create a vector with samples in the right order,
# according to the "Timepoint" column (from smallest to greatest):
sample_order <- order(immdata$meta$Timepoint)
# Sanity check: timepoints are following the right order:
immdata$meta$Timepoint[sample_order]
# Samples, sorted by the timepoints:
immdata$meta$Sample[sample_order]
# And finally, we visualise the data:
vis(tc, .order = sample_order)
```

---

vis — *One function to visualise them all*

Description

Output from every function in immunarch can be visualised with a single function - `vis`. The `vis` automatically detects the type of the data and draws a proper visualisation. For example, output from the `repOverlap` function will be identified as repertoire overlap values and respective visualisation will be chosen without any additional arguments. See "Details" for the list of available visualisations.

Usage

```
vis(.data, ...)
```

Arguments

`.data`	Pass the output from any immunarch analysis tool to `vis()`.
`...`	Any other arguments, see the "Details" section for specific visualisation functions.

Details

List of available visualisations for different kinds of data.

Basic analysis:

- Exploratory analysis results (from repExplore) - see vis.immunr_exp_vol;

- Clonality statistics (from repClonality) - see vis.immunr_homeo.

Overlaps and public clonotypes:

- Overlaps (from repOverlap) using heatmaps, circos plots, polar area plots - see vis.immunr_ov_matrix;

- Overlap clustering (from repOverlapAnalysis) - see vis.immunr_hclust;

- Repertoire incremental overlaps (from repOverlap) - see vis.immunr_inc_overlap;

- Public repertoire abundance (from pubRep) - vis vis.immunr_public_repertoire.

Gene usage:

- Gene usage statistics (from geneUsage) using bar plots, box plots - see vis.immunr_gene_usage;

- Gene usage distances (from geneUsageAnalysis) using heatmaps, circos plots, polar area plots - see vis.immunr_ov_matrix;

- Gene usage clustering (from geneUsageAnalysis) - see vis.immunr_hclust.

Diversity estimation:

- Diversity estimations (from repDiversity) - see vis.immunr_chao1.

BCR analysis:

- Clonal tree (from repClonalFamily) - see vis.clonal_family and vis.clonal_family_tree.

Advanced analysis:

- Repertoire dynamics (from trackClonotypes) - see vis.immunr_dynamics;

- Sequence logo plots of amino acid distributions (from kmer_profile) - see vis_seqlogo;

- Kmers distributions (from getKmers) - see vis.immunr_kmer_table;

- Mutation networks (from mutationNetwork) - Work In Progress on vis.immunr_mutation_network;

- CDR3 amino acid properties, e.g., biophysical (from cdrProp) - Work In Progress on vis.immunr_cdr_prop.

Additionaly, we provide a wrapper functions for visualisations of common data types:

- Any data frames or matrices using heatmaps - see vis_heatmap and vis_heatmap2;

- Any data frames or matrices using circos plots - see vis_circos.

Value

A ggplot2, pheatmap or circlize object.

See Also

fixVis for precise manipulation of plots.

Examples

```
# Load the test data
data(immdata)

# Compute and visualise:
ov <- repOverlap(immdata$data)
vis(ov)

gu <- geneUsage(immdata$data)
vis(gu)
```

```
dv <- repDiversity(immdata$data)
vis(dv)
```

vis.clonal_family *Visualise clonal family tree: wrapper for calling on the entire repClonalFamily output*

Description

Visualise clonal family tree: wrapper for calling on the entire repClonalFamily output

Usage

```
## S3 method for class 'clonal_family'
vis(.data, ...)
```

Arguments

`.data`	Clonal families from 1 or multiple samples: `repClonalFamily` output.
`...`	Not used here.

Value

A ggraph object.

Examples

```
data(bcrdata)
bcr_data <- bcrdata$data

clonal_family <- bcr_data %>%
  seqCluster(seqDist(bcr_data), .fixed_threshold = 3) %>%
  repGermline(.threads = 1) %>%
  repAlignLineage(.min_lineage_sequences = 2, .align_threads = 2, .nofail = TRUE) %>%
  repClonalFamily(.threads = 1, .nofail = TRUE) %>%
  vis()
```

vis.clonal_family_tree

Visualise clonal family tree

Description

Visualise clonal family tree

Usage

```
## S3 method for class 'clonal_family_tree'
vis(.data, ...)
```

Arguments

`.data`	Single clonal family tree data from 1 cluster: 1 element from TreeStats column from `repClonalFamily` output.
`...`	Not used here.

Value

A ggraph object.

Examples

```
data(bcrdata)
bcr_data <- bcrdata$data

clonal_family <- bcr_data %>%
  seqCluster(seqDist(bcr_data), .fixed_threshold = 3) %>%
  repGermline(.threads = 1) %>%
  repAlignLineage(.min_lineage_sequences = 2, .align_threads = 2, .nofail = TRUE) %>%
  repClonalFamily(.threads = 1, .nofail = TRUE)

# This condition can be omitted; it prevents the example from crashing
# when ClustalW or PHYLIP are not installed
if (!("step_failure_ignored" %in% class(clonal_family))) {
  vis(clonal_family[["full_clones"]][["TreeStats"]][[2]])
}
```

vis.immunr_chao1 *Visualise diversity.*

Description

An utility function to visualise the output from repDiversity.

Usage

```
## S3 method for class 'immunr_chao1'
vis(
  .data,
  .by = NA,
  .meta = NA,
  .errorbars = c(0.025, 0.975),
  .errorbars.off = FALSE,
  .points = TRUE,
  .test = TRUE,
  .signif.label.size = 3.5,
  ...
)
```

Arguments

`.data` Output from repDiversity.

`.by` Pass NA if you want to plot samples without grouping.

You can pass a character vector with one or several column names from ".meta" to group your data before plotting. In this case you should provide ".meta".

You can pass a character vector that exactly matches the number of samples in your data, each value should correspond to a sample's property. It will be used to group data based on the values provided. Note that in this case you should pass NA to ".meta".

`.meta` A metadata object. An R dataframe with sample names and their properties, such as age, serostatus or hla.

`.errorbars` A numeric vector of length two with quantiles for error bars on sectors. Disabled if ".errorbars.off" is TRUE.

`.errorbars.off` If TRUE then plot CI bars for distances between each group. Disabled if no group passed to the ".by" argument.

`.points` A logical value defining whether points will be visualised or not.

`.test` A logical vector whether statistical tests should be applied. See "Details" for more information.

`.signif.label.size` An integer value defining the size of text for p-value.

`...` Not used here.

Details

If data is grouped, then statistical tests for comparing means of groups will be performed, unless .test = FALSE is supplied. In case there are only two groups, the Wilcoxon rank sum test (https://en.wikipedia.org/wiki/Wilcoxon_signed-rank_test) is performed (R function wilcox.test with an argument exact = FALSE) for testing if there is a difference in mean rank values between two groups. In case there more than two groups, the Kruskal-Wallis test (https://en.wikipedia.org/wiki/Kruskal A significant Kruskal-Wallis test indicates that at least one sample stochastically dominates one other sample. Adjusted for multiple comparisons P-values are plotted on the top of groups. P-value adjusting is done using the Holm method (https://en.wikipedia.org/wiki/Holm You can execute the command ?p.adjust in the R console to see more.

Value

A ggplot2 object.

See Also

repDiversity vis

Examples

```
data(immdata)
dv <- repDiversity(immdata$data, "chao1")
vis(dv)
```

---

vis.immunr_clonal_prop

Visualise results of the clonality analysis

Description

An utility function to visualise the output from repClonality.

Usage

```
## S3 method for class 'immunr_clonal_prop'
vis(
  .data,
  .by = NA,
  .meta = NA,
  .errorbars = c(0.025, 0.975),
  .errorbars.off = FALSE,
  .points = TRUE,
  .test = TRUE,
  .signif.label.size = 3.5,
  ...
)
```

Arguments

`.data` Output from `repClonality`.

`.by` Pass NA if you want to plot samples without grouping.

You can pass a character vector with one or several column names from ".meta" to group your data before plotting. In this case you should provide ".meta".

You can pass a character vector that exactly matches the number of samples in your data, each value should correspond to a sample's property. It will be used to group data based on the values provided. Note that in this case you should pass NA to ".meta".

`.meta` A metadata object. An R dataframe with sample names and their properties, such as age, serostatus or hla.

`.errorbars` A numeric vector of length two with quantiles for error bars on sectors. Disabled if ".errorbars.off" is TRUE.

`.errorbars.off` If TRUE then plot CI bars for distances between each group. Disabled if no group passed to the ".by" argument.

`.points` A logical value defining whether points will be visualised or not.

`.test` A logical vector whether statistical tests should be applied. See "Details" for more information.

`.signif.label.size`

An integer value defining the size of text for p-value.

`...` Not used here.

Details

If data is grouped, then statistical tests for comparing means of groups will be performed, unless `.test = FALSE` is supplied. In case there are only two groups, the Wilcoxon rank sum test (https://en.wikipedia.org/wiki/Wilcoxon_signed-rank_test) is performed (R function `wilcox.test` with an argument `exact = FALSE`) for testing if there is a difference in mean rank values between two groups. In case there more than two groups, the Kruskal-Wallis test (https://en.wikipedia.org/wiki/Kruskal A significant Kruskal-Wallis test indicates that at least one sample stochastically dominates one other sample. Adjusted for multiple comparisons P-values are plotted on the top of groups. P-value adjusting is done using the Holm method (https://en.wikipedia.org/wiki/Holm You can execute the command `?p.adjust` in the R console to see more.

Value

A ggplot2 object.

See Also

repClonality vis

Examples

```
data(immdata)
clp <- repClonality(immdata$data, "clonal.prop")
```

```
vis(clp)

hom <- repClonality(immdata$data, "homeo")
# Remove p values and points from the plot
vis(hom, .by = "Status", .meta = immdata$meta, .test = FALSE, .points = FALSE)
```

`vis.immunr_dynamics` *Visualise clonotype dynamics*

Description

Visualise clonotype dynamics

Usage

```
## S3 method for class 'immunr_dynamics'
vis(.data, .plot = c("smooth", "area", "line"), .order = NA, .log = FALSE, ...)
```

Arguments

`.data`	Output from the trackClonotypes function.
`.plot`	Character. Either "smooth", "area" or "line". Each specifies a type of plot for visualisation of clonotype dynamics.
`.order`	Numeric or character vector. Specifies the order to samples, e.g., it used for ordering samples by timepoints. Either See "Examples" below for more details.
`.log`	Logical. If TRUE then use log-scale for the frequency axis.
`...`	Not used here.

Value

A ggplot2 object.

Examples

```
# Load an example data that comes with immunarch
data(immdata)

# Make the data smaller in order to speed up the examples
immdata$data <- immdata$data[c(1, 2, 3, 7, 8, 9)]
immdata$meta <- immdata$meta[c(1, 2, 3, 7, 8, 9), ]

# Option 1
# Choose the first 10 amino acid clonotype sequences
# from the first repertoire to track
tc <- trackClonotypes(immdata$data, list(1, 10), .col = "aa")
# Choose the first 20 nucleotide clonotype sequences
# and their V genes from the "MS1" repertoire to track
tc <- trackClonotypes(immdata$data, list("MS1", 20), .col = "nt+v")
```

```
# Option 2
# Choose clonotypes with amino acid sequences "CASRGLITDTQYF" or "CSASRGSPNEQYF"
tc <- trackClonotypes(immdata$data, c("CASRGLITDTQYF", "CSASRGSPNEQYF"), .col = "aa")

# Option 3
# Choose the first 10 clonotypes from the first repertoire
# with amino acid sequences and V segments
target <- immdata$data[[1]] %>%
  select(CDR3.aa, V.name) %>%
  head(10)
tc <- trackClonotypes(immdata$data, target)

# Visualise the output regardless of the chosen option
# Therea are three way to visualise it, regulated by the .plot argument
vis(tc, .plot = "smooth")
vis(tc, .plot = "area")
vis(tc, .plot = "line")

# Visualising timepoints
# First, we create an additional column in the metadata with randomly choosen timepoints:
immdata$meta$Timepoint <- sample(1:length(immdata$data))
immdata$meta
# Next, we create a vector with samples in the right order,
# according to the "Timepoint" column (from smallest to greatest):
sample_order <- order(immdata$meta$Timepoint)
# Sanity check: timepoints are following the right order:
immdata$meta$Timepoint[sample_order]
# Samples, sorted by the timepoints:
immdata$meta$Sample[sample_order]
# And finally, we visualise the data:
vis(tc, .order = sample_order)
```

vis.immunr_exp_vol *Visualise results of the exploratory analysis*

Description

An utility function to visualise the output from repExplore.

Usage

```
## S3 method for class 'immunr_exp_vol'
vis(
  .data,
  .by = NA,
  .meta = NA,
  .errorbars = c(0.025, 0.975),
  .errorbars.off = FALSE,
  .points = TRUE,
  .test = TRUE,
  .signif.label.size = 3.5,
  ...
)
```

Arguments

`.data` Output from `repExplore`.

`.by` Pass NA if you want to plot samples without grouping.

You can pass a character vector with one or several column names from ".meta" to group your data before plotting. In this case you should provide ".meta".

You can pass a character vector that exactly matches the number of samples in your data, each value should correspond to a sample's property. It will be used to group data based on the values provided. Note that in this case you should pass NA to ".meta".

`.meta` A metadata object. An R dataframe with sample names and their properties, such as age, serostatus or hla.

`.errorbars` A numeric vector of length two with quantiles for error bars on sectors. Disabled if ".errorbars.off" is TRUE.

`.errorbars.off` If TRUE then plot CI bars for distances between each group. Disabled if no group passed to the ".by" argument.

`.points` A logical value defining whether points will be visualised or not.

`.test` A logical vector whether statistical tests should be applied. See "Details" for more information.

`.signif.label.size`

An integer value defining the size of text for p-value.

`...` Not used here.

Details

If data is grouped, then statistical tests for comparing means of groups will be performed, unless `.test = FALSE` is supplied. In case there are only two groups, the Wilcoxon rank sum test (https://en.wikipedia.org/wiki/Wilcoxon_signed-rank_test) is performed (R function `wilcox.test` with an argument `exact = FALSE`) for testing if there is a difference in mean rank values between two groups. In case there more than two groups, the Kruskal-Wallis test (https://en.wikipedia.org/wiki/Kruskal A significant Kruskal-Wallis test indicates that at least one sample stochastically dominates one other sample. Adjusted for multiple comparisons P-values are plotted on the top of groups. P-value adjusting is done using the Holm method (https://en.wikipedia.org/wiki/Holm You can execute the command `?p.adjust` in the R console to see more.

Value

A ggplot2 object.

See Also

repExplore vis

Examples

```
data(immdata)
repExplore(immdata$data, "volume") %>% vis()
repExplore(immdata$data, "count") %>% vis()
repExplore(immdata$data, "len") %>% vis()
repExplore(immdata$data, "clones") %>% vis()
```

`vis.immunr_gene_usage` *Histograms and boxplots (general case / gene usage)*

Description

Visualise distributions of genes using heatmaps or other plots.

Usage

```
## S3 method for class 'immunr_gene_usage'
vis(.data, .plot = c("hist", "box", "heatmap", "heatmap2", "circos"), ...)
```

Arguments

`.data`	Output from the geneUsage function.
`.plot`	String specifying the plot type: - "hist" for histograms using vis_hist; - "heatmap" for heatmaps using vis_heatmap; - "heatmap2" for heatmaps using vis_heatmap2; - "circos" for circos plots using vis_circos.
`...`	Other arguments passed to corresponding functions depending on the plot type: - "hist" - passes arguments to vis_hist; - "box" - passes arguments to vis_box; - "heatmap" - passes arguments to vis_heatmap; - "heatmap2" - passes arguments to vis_heatmap2 and heatmap from the "pheatmap" package; - "circos" - passes arguments to vis_circos and chordDiagram from the "circlize" package.

Value

A ggplot2 object, pheatmap or circlize object.

See Also

geneUsage

Examples

```
data(immdata)

gu <- geneUsage(immdata$data[[1]])
vis(gu)

gu <- geneUsage(immdata$data)
vis(gu, .by = "Status", .meta = immdata$meta)
vis(gu, "box", .by = "Status", .meta = immdata$meta)
```

---

vis.immunr_hclust *Visualisation of hierarchical clustering*

Description

Visualisation of the results of hierarchical clustering. For other clustering visualisations see vis.immunr_kmeans.

Usage

```
## S3 method for class 'immunr_hclust'
vis(.data, .rect = FALSE, .plot = c("clust", "best"), ...)
```

Arguments

.data	Clustering results from repOverlapAnalysis or geneUsageAnalysis.
.rect	Passed to fviz_dend - whether to add a rectangle around groups.
.plot	A character vector of length one or two specifying which plots to visualise. If "clust" then plot only the clustering. If "best" then plot the number of optimal clusters. If both then plot both.
...	Not used here.

Value

Ggplot2 objects inside the patchwork container.

See Also

vis, repOverlapAnalysis, geneUsageAnalysis

Examples

```
data(immdata)
ov <- repOverlap(immdata$data)
repOverlapAnalysis(ov, "mds+hclust") %>% vis()
```

vis.immunr_inc_overlap

Visualise incremental overlaps

Description

Visualise incremental overlaps

Usage

```
## S3 method for class 'immunr_inc_overlap'
vis(.data, .target = 1, .grid = FALSE, .ncol = 2, ...)
```

Arguments

`.data`	Output from the repOverlap function that uses "top" methods.
`.target`	Index of a repertoire to plot. Omitted if .grid is TRUE.
`.grid`	Logical. If TRUE then plot all similarities in a grid.
`.ncol`	Numeric. Number of columns in the resulting grid.
`...`	Not used here.

Value

A ggplot2 object.

See Also

repOverlap

Examples

```
data(immdata)
tmp <- repOverlap(immdata$data[1:4], "inc+overlap", .verbose.inc = FALSE, .verbose = FALSE)
vis(tmp, .target = 1)
vis(tmp, .grid = TRUE)
```

vis.immunr_kmeans *Visualisation of K-means and DBSCAN clustering*

Description

Visualisation of the results of K-means and DBSCAN clustering. For hierarhical clustering visualisations see vis.immunr_hclust.

Usage

```
## S3 method for class 'immunr_kmeans'
vis(
  .data,
  .point = TRUE,
  .text = TRUE,
  .ellipse = TRUE,
  .point.size = 2,
  .text.size = 10,
  .plot = c("clust", "best"),
  ...
)
```

Arguments

`.data`	Clustering results from repOverlapAnalysis or geneUsageAnalysis.
`.point`	If TRUE then plot sample points. Passed to fviz_cluster.
`.text`	If TRUE then plot text labels. Passed to fviz_cluster.
`.ellipse`	If TRUE then plot ellipses around all samples. Passed to "ellipse" from fviz_cluster.
`.point.size`	Size of points, passed to "pointsize" from fviz_cluster.
`.text.size`	Size of text labels, passed to labelsize from fviz_cluster.
`.plot`	A character vector of length one or two specifying which plots to visualise. If "clust" then plot only the clustering. If "best" then plot the number of optimal clusters. If both then plot both.
`...`	Not used here.

Value

Ggplot2 objects inside the pathwork container.

See Also

vis, repOverlapAnalysis, geneUsageAnalysis

Examples

```
data(immdata)
ov <- repOverlap(immdata$data)
repOverlapAnalysis(ov, "mds+kmeans") %>% vis()
```

---

vis.immunr_kmer_table *Most frequent kmers visualisation.*

Description

Plot a distribution (bar plot) of the most frequent kmers in a data.

Usage

```
## S3 method for class 'immunr_kmer_table'
vis(
  .data,
  .head = 100,
  .position = c("stack", "dodge", "fill"),
  .log = FALSE,
  ...
)
```

Arguments

`.data`	Data frame with two columns "Kmers" and "Count" or a list with such data frames. See Examples.
`.head`	Number of the most frequent kmers to choose for plotting from each data frame.
`.position`	Character vector of length 1. Position of bars for each kmers. Value for the `ggplot2` argument `position`.
`.log`	Logical. If TRUE then plot log-scaled plots.
`...`	Not used here.

Value

A ggplot2 object.

See Also

`get.kmers`

Examples

```
# Load necessary data and package.
data(immdata)
# Get 5-mers.
imm.km <- getKmers(immdata$data[[1]], 5)
# Plots for kmer proportions in each data frame in immdata.
p1 <- vis(imm.km, .position = "stack")
p2 <- vis(imm.km, .position = "fill")
p1 + p2
```

---

vis.immunr_mds *PCA / MDS / tSNE visualisation (mainly overlap / gene usage)*

Description

PCA / MDS / tSNE visualisation (mainly overlap / gene usage)

Usage

```
## S3 method for class 'immunr_mds'
vis(
  .data,
  .by = NA,
  .meta = NA,
  .point = TRUE,
  .text = TRUE,
  .ellipse = TRUE,
  .point.size = 2,
  .text.size = 4,
  ...
)
```

Arguments

.data	Output from analysis functions such as geneUsageAnalysis or immunr_pca, immunr_mds or immunr_tsne.
.by	Pass NA if you want to plot samples without grouping. You can pass a character vector with one or several column names from ".meta" to group your data before plotting. In this case you should provide ".meta". You can pass a character vector that exactly matches the number of samples in your data, each value should correspond to a sample's property. It will be used to group data based on the values provided. Note that in this case you should pass NA to ".meta".
.meta	A metadata object. An R dataframe with sample names and their properties, such as age, serostatus or hla.
.point	Logical. If TRUE then plot points corresponding to objects.

`.text` Logical. If TRUE then plot sample names.

`.ellipse` Logical. If TRUE then plot ellipses around clusters of grouped samples.

`.point.size` Numeric. A size of points to plot.

`.text.size` Numeric. A size of sample names' labels.

`...` Not used here.

Details

Other visualisation methods:

- PCA - vis.immunr_pca

- MDS - vis.immunr_mds

- tSNE - vis.immunr_tsne

Value

A ggplot2 object.

Examples

```
data(immdata)
ov <- repOverlap(immdata$data)
repOverlapAnalysis(ov, "mds") %>% vis()
```

`vis.immunr_ov_matrix` *Repertoire overlap and gene usage visualisations*

Description

Visualises matrices with overlap values or gene usage distances among samples. For details see the links below.

Usage

```
## S3 method for class 'immunr_ov_matrix'
vis(.data, .plot = c("heatmap", "heatmap2", "circos"), ...)
```

Arguments

`.data` Output from repOverlap or geneUsageAnalysis.

`.plot` A string specifying the plot type:
- "heatmap" for heatmaps using vis_heatmap;
- "heatmap2" for heatmaps using vis_heatmap2;
- "circos" for circos plots using vis_circos;

`...` Other arguments are passed through to the underlying plotting function:

- "heatmap" - passes arguments to vis_heatmap;

- "heatmap2" - passes arguments to vis_heatmap2 and heatmap from the "pheatmap" package;

- "circos" - passes arguments to vis_circos and chordDiagram from the "circlize" package;

Value

A ggplot2, pheatmap or circlize object.

Examples

```
data(immdata)
ov <- repOverlap(immdata$data)
vis(ov)
vis(ov, "heatmap")
vis(ov, "heatmap2")
vis(ov, "circos")
```

vis.immunr_public_repertoire

Public repertoire visualisation

Description

Public repertoire visualisation

Usage

```
## S3 method for class 'immunr_public_repertoire'
vis(.data, .plot = c("freq", "clonotypes"), ...)
```

Arguments

`.data` Public repertoire, an output from pubRep.

`.plot` A string specifying the plot type:

- "freq" for visualisation of the distribution of occurrences of clonotypes and their frequencies using vis_public_frequencies.

- "clonotypes" for visualisation of public clonotype frequenciy correlations between pairs of samples using vis_public_clonotypes

`...` Further arguments passed vis_public_frequencies or vis_public_clonotypes, depending on the ".plot" argument.

Value

A ggplot2 object.

Examples

```
data(immdata)
immdata$data <- lapply(immdata$data, head, 300)
pr <- pubRep(immdata$data, .verbose = FALSE)
vis(pr, "freq")
vis(pr, "freq", .type = "none")

vis(pr, "clonotypes", 1, 2)
```

---

vis.immunr_public_statistics

Visualise sharing of clonotypes among samples

Description

Visualise public clonotype frequencies.

Usage

```
## S3 method for class 'immunr_public_statistics'
vis(.data, ...)
```

Arguments

.data	Public repertoire - an output from the pubRep function.
...	Other arguments passsed directly to upset.

Value

A ggplot2 object.

Examples

```
data(immdata)
immdata$data <- lapply(immdata$data, head, 2000)
pr <- pubRep(immdata$data, .verbose = FALSE)
pubRepStatistics(pr) %>% vis()
```

vis.step_failure_ignored

Handler for .nofail argument of pipeline steps that prevents examples from crashing on computers where certain dependencies are not installed

Description

Handler for .nofail argument of pipeline steps that prevents examples from crashing on computers where certain dependencies are not installed

Usage

```
## S3 method for class 'step_failure_ignored'
vis(.data, ...)
```

Arguments

.data	Not used here.
...	Not used here.

Value

An empty object with "step_failure_ignored" class.

vis_bar

Bar plots

Description

Bar plots

Usage

```
vis_bar(
  .data,
  .by = NA,
  .meta = NA,
  .errorbars = c(0.025, 0.975),
  .errorbars.off = FALSE,
  .stack = FALSE,
  .points = TRUE,
  .test = TRUE,
  .signif.label.size = 3.5,
  .errorbar.width = 0.2,
  .defgroupby = "Sample",
  .grouping.var = "Group",
  .labs = c("X", "Y"),
  .title = "Barplot (.title argument)",
  .subtitle = "Subtitle (.subtitle argument)",
  .legend = NA,
  .leg.title = "Legend (.leg.title argument)",
  .legend.pos = "right",
  .rotate_x = 90
)
```

Arguments

`.data` Data to visualise.

`.by` Pass NA if you want to plot samples without grouping.

You can pass a character vector with one or several column names from ".meta" to group your data before plotting. In this case you should provide ".meta".

You can pass a character vector that exactly matches the number of samples in your data, each value should correspond to a sample's property. It will be used to group data based on the values provided. Note that in this case you should pass NA to ".meta".

`.meta` A metadata object. An R dataframe with sample names and their properties, such as age, serostatus or hla.

`.errorbars` A numeric vector of length two with quantiles for error bars on sectors. Disabled if ".errorbars.off" is TRUE.

`.errorbars.off` If TRUE then plot CI bars for distances between each group. Disabled if no group passed to the ".by" argument.

`.stack` If TRUE and .errorbars.off is TRUE then plot stacked bar plots for each Group or Sample

`.points` A logical value defining whether points will be visualised or not.

`.test` A logical vector whether statistical tests should be applied. See "Details" for more information.

`.signif.label.size`

An integer value defining the size of text for p-value.

`.errorbar.width`

Numeric. Width for error bars.

`.defgroupby` A name for the column with sample names.

`.grouping.var` A name for the column to group by.

`.labs` A character vector of length two specifying names for x-axis and y-axis.

`.title` The text for the plot's title.

`.subtitle` The text for the plot's subtitle.

`.legend` If TRUE then displays a legend, otherwise removes legend from the plot.

`.leg.title` The text for the plots's legend. Provide NULL to remove the legend's title completely.

`.legend.pos` Positions of the legend: either "top", "bottom", "left" or "right".

`.rotate_x` How much the x tick text should be rotated? In angles.

Value

A ggplot2 object.

Examples

```
vis_bar(data.frame(Sample = c("A", "B", "C"), Value = c(1, 2, 3)))
```

---

vis_box *Flexible box-plots for visualisation of distributions*

Description

Visualisation of distributions using ggplot2-based boxplots.

Usage

```
vis_box(
  .data,
  .by = NA,
  .meta = NA,
  .melt = TRUE,
  .points = TRUE,
  .test = TRUE,
  .signif.label.size = 3.5,
  .defgroupby = "Sample",
  .grouping.var = "Group",
  .labs = c("X", "Y"),
  .title = "Boxplot (.title argument)",
  .subtitle = "Subtitle (.subtitle argument)",
  .legend = NA,
  .leg.title = "Legend (.leg.title argument)",
  .legend.pos = "right"
)
```

Arguments

`.data` Input matrix or data frame.

`.by` Pass NA if you want to plot samples without grouping.

You can pass a character vector with one or several column names from ".meta" to group your data before plotting. In this case you should provide ".meta".

You can pass a character vector that exactly matches the number of samples in your data, each value should correspond to a sample's property. It will be used to group data based on the values provided. Note that in this case you should pass NA to ".meta".

`.meta` A metadata object. An R dataframe with sample names and their properties, such as age, serostatus or hla.

`.melt` If TRUE then apply melt to the ".data" before plotting. In this case ".data" is supposed to be a data frame with the first character column reserved for names of genes and other numeric columns reserved to counts or frequencies of genes. Each numeric column should be associated with a specific repertoire sample.

`.points` A logical value defining whether points will be visualised or not.

`.test` A logical vector whether statistical tests should be applied. See "Details" for more information.

`.signif.label.size` An integer value defining the size of text for p-value.

`.defgroupby` A name for the column with sample names.

`.grouping.var` A name for the column to group by.

`.labs` Character vector of length two with names for x-axis and y-axis, respectively.

`.title` The text for the title of the plot.

`.subtitle` The The text for the plot's subtitle.

`.legend` If TRUE then displays a legend, otherwise removes legend from the plot.

`.leg.title` The The text for the plots's legend. Provide NULL to remove the legend's title completely.

`.legend.pos` Positions of the legend: either "top", "bottom", "left" or "right".

Value

A ggplot2 object.

See Also

vis.immunr_gene_usage, geneUsage

Examples

```
vis_box(data.frame(Sample = sample(c("A", "B", "C"), 100, TRUE), Value = rnorm(100)), .melt = FALSE)
```

---

vis_circos *Visualisation of matrices using circos plots*

Description

Visualise matrices with the chordDiagram function from the circlize package.

Usage

```
vis_circos(.data, .title = NULL, ...)
```

Arguments

`.data`	Input matrix.
`.title`	The The text for the title of the plot.
`...`	Other arguments passed to chordDiagram from the 'circlize' package.

Value

A circlize object.

See Also

vis, repOverlap.

Examples

```
data(immdata)
ov <- repOverlap(immdata$data)
vis(ov, .plot = "circos")
```

`vis_heatmap` *Visualisation of matrices and data frames using ggplo2-based heatmaps*

Description

Fast and easy visualisations of matrices or data frames with functions based on the ggplot2 package.

Usage

```
vis_heatmap(
  .data,
  .text = TRUE,
  .scientific = FALSE,
  .signif.digits = 2,
  .text.size = 4,
  .axis.text.size = NULL,
  .labs = c("Sample", "Sample"),
  .title = "Overlap",
  .leg.title = "Overlap values",
  .legend = TRUE,
  .na.value = NA,
  .transpose = FALSE,
  ...
)
```

Arguments

`.data` Input object: a matrix or a data frame.

If matrix: column names and row names (if presented) will be used as names for labs.

If data frame: the first column will be used for row names and removed from the data. Other columns will be used for values in the heatmap.

`.text` If TRUE then plots values in the heatmap cells. If FALSE does not plot values, just plot coloured cells instead.

`.scientific` If TRUE then uses the scientific notation for numbers (e.g., "2.0e+2").

`.signif.digits` Number of significant digits to display on plot.

`.text.size` Size of text in the cells of heatmap.

`.axis.text.size`

Size of text on the axis labels.

`.labs` A character vector of length two with names for x-axis and y-axis, respectively.

`.title` The The text for the plot’s title.

`.leg.title` The The text for the plots’s legend. Provide NULL to remove the legend’s title completely.

`.legend` If TRUE then displays a legend, otherwise removes the legend from the plot.

`.na.value` Replace NA values with this value. By default they remain NA.

`.transpose` Logical. If TRUE then switch rows and columns.

`...` Other passed arguments.

Value

A ggplot2 object.

See Also

vis, repOverlap.

Examples

```
data(immdata)
ov <- repOverlap(immdata$data)
vis_heatmap(ov)
gu <- geneUsage(immdata$data, "hs.trbj")
vis_heatmap(gu)
```

vis_heatmap2 *Visualisation of matrices using pheatmap-based heatmaps*

Description

Visualise matrices with the functions based on the pheatmap package with minimum amount of arguments.

Usage

```
vis_heatmap2(
  .data,
  .meta = NA,
  .by = NA,
  .title = NA,
  .color = colorRampPalette(c("#67001f", "#d6604d", "#f7f7f7", "#4393c3",
    "#053061"))(1024),
  ...
)
```

Arguments

.data	Input matrix. Column names and row names (if presented) will be used as names for labs.
.meta	A metadata object. An R dataframe with sample names and their properties, such as age, serostatus or hla.
.by	Set NA if you want to plot samples without grouping.
.title	The text for the plot's title (same as the "main" argument in pheatmap).
.color	A vector specifying the colors (same as the "color" argument in pheatmap). Pass NA to use the default pheatmap colors.
...	Other arguments for the pheatmap function.

Value

A pheatmap object.

See Also

vis, repOverlap

Examples

```
data(immdata)
ov <- repOverlap(immdata$data)
vis_heatmap2(ov)
```

vis_hist *Visualisation of distributions using histograms*

Description

Visualisation of distributions using ggplot2-based histograms.

Usage

```
vis_hist(
  .data,
  .by = NA,
  .meta = NA,
  .title = "Gene usage",
  .ncol = NA,
  .points = TRUE,
  .test = TRUE,
  .coord.flip = FALSE,
  .grid = FALSE,
  .labs = c("Gene", NA),
  .melt = TRUE,
  .legend = NA,
  .add.layer = NULL,
  ...
)
```

Arguments

`.data`	Input matrix or data frame.
`.by`	Pass NA if you want to plot samples without grouping. You can pass a character vector with one or several column names from ".meta" to group your data before plotting. In this case you should provide ".meta". You can pass a character vector that exactly matches the number of samples in your data, each value should correspond to a sample's property. It will be used to group data based on the values provided. Note that in this case you should pass NA to ".meta".
`.meta`	A metadata object. An R dataframe with sample names and their properties, such as age, serostatus or hla.
`.title`	The text for the title of the plot.
`.ncol`	A number of columns to display. Provide NA (by default) if you want the function to automatically detect the optimal number of columns.
`.points`	A logical value defining whether points will be visualised or not.
`.test`	A logical vector whether statistical tests should be applied. See "Details" for more information.
`.coord.flip`	If TRUE then swap x- and y-axes.

`.grid` If TRUE then plot separate visualisations for each sample.

`.labs` A character vector of length two with names for x-axis and y-axis, respectively.

`.melt` If TRUE then apply melt to the ".data" before plotting. In this case ".data" is supposed to be a data frame with the first character column reserved for names of genes and other numeric columns reserved to counts or frequencies of genes. Each numeric column should be associated with a specific repertoire sample.

`.legend` If TRUE then plots the legend. If FALSE removes the legend from the plot. If NA automatically detects the best way to display legend.

`.add.layer` Addditional ggplot2 layers, that added to each plot in the output plot or grid of plots.

`...` Is not used here.

Details

If data is grouped, then statistical tests for comparing means of groups will be performed, unless `.test = FALSE` is supplied. In case there are only two groups, the Wilcoxon rank sum test (https://en.wikipedia.org/wiki/Wilcoxon_signed-rank_test) is performed (R function `wilcox.test` with an argument `exact = FALSE`) for testing if there is a difference in mean rank values between two groups. In case there more than two groups, the Kruskal-Wallis test (https://en.wikipedia.org/wiki/Kruskal A significant Kruskal-Wallis test indicates that at least one sample stochastically dominates one other sample. Adjusted for multiple comparisons P-values are plotted on the top of groups. P-value adjusting is done using the Holm method (https://en.wikipedia.org/wiki/Holm You can execute the command `?p.adjust` in the R console to see more.

Value

A ggplot2 object.

See Also

vis.immunr_gene_usage, geneUsage

Examples

```
data(immdata)
imm_gu <- geneUsage(immdata$data[[1]])
vis(imm_gu,
  .plot = "hist", .add.layer =
    theme(axis.text.x = element_text(angle = 75, vjust = 1))
)
imm_gu <- geneUsage(immdata$data[1:4])
vis(imm_gu,
  .plot = "hist", .grid = TRUE, .add.layer =
    theme(axis.text.x = element_text(angle = 75, vjust = 1))
)
```

vis_immunr_kmer_profile_main *Visualise kmer profiles*

Description

Visualise kmer profiles

Usage

```
vis_immunr_kmer_profile_main(.data, .plot, ...)
```

Arguments

`.data`	Kmer data, an output from kmer_profile.
`.plot`	String specifying the plot type: - "seqlogo" for traditional sequence logo plots using vis_seqlogo; - "textlogo" for modified approach to sequence logo plots via text labels using vis_textlogo;
`...`	Other arguments passed to vis_textlogo or vis_seqlogo, depending on the ".plot" argument.

Value

A ggplot2 object.

Examples

```
data(immdata)
getKmers(immdata$data[[1]], 5) %>%
  kmer_profile() %>%
  vis("seqlogo")
```

vis_public_clonotypes *Visualisation of public clonotypes*

Description

Visualise correlation of public clonotype frequencies in pairs of repertoires.

Usage

```
vis_public_clonotypes(
  .data,
  .x.rep = NA,
  .y.rep = NA,
  .title = NA,
  .ncol = 3,
  .point.size.modif = 1,
  .cut.axes = TRUE,
  .density = TRUE,
  .lm = TRUE,
  .radj.size = 3.5
)
```

Arguments

`.data` Public repertoire data - an output from the pubRep function.

`.x.rep` Either indices of samples or character vector of sample names for the x-axis. Must be of the same length as ".y.rep".

`.y.rep` Either indices of samples or character vector of sample names for the y-axis. Must be of the same length as ".x.rep".

`.title` The text for the title of the plot.

`.ncol` An integer number of columns to print in the grid of pairs of repertoires.

`.point.size.modif` An integer value that is a modifier of the point size. The larger the number, the larger the points.

`.cut.axes` If TRUE then axes limits become shorter.

`.density` If TRUE then displays density plot for distributions of clonotypes for each sample. If FALSE then removes density plot from the visualisation.

`.lm` If TRUE then fit a linear model and displays an R adjusted coefficient that shows how similar samples are in terms of shared clonotypes.

`.radj.size` An integer value, that defines the size of the The text for the R adjusted coefficient.

Value

A ggplot2 object.

See Also

pubRep, vis.immunr_public_repertoire

Examples

```
data(immdata)
pr <- pubRep(immdata$data, .verbose = FALSE)
vis(pr, "clonotypes", 1, 2)
```

vis_public_frequencies *Public repertoire visualisation*

Description

Visualise public clonotype frequencies.

Usage

```
vis_public_frequencies(
  .data,
  .by = NA,
  .meta = NA,
  .type = c("boxplot", "none", "mean")
)
```

Arguments

.data	Public repertoire - an output from the pubRep function.
.by	Pass NA if you want to plot samples without grouping. You can pass a character vector with one or several column names from ".meta" to group your data before plotting. In this case you should provide ".meta". You can pass a character vector that exactly matches the number of samples in your data, each value should correspond to a sample's property. It will be used to group data based on the values provided. Note that in this case you should pass NA to ".meta".
.meta	A metadata object. An R dataframe with sample names and their properties, such as age, serostatus or hla.
.type	Character. Either "boxplot" for plotting distributions of frequencies, "none" for plotting everything, or "mean" for plotting average values only.

Value

A ggplot2 object.

Examples

```
data(immdata)
immdata$data <- lapply(immdata$data, head, 500)
pr <- pubRep(immdata$data, .verbose = FALSE)
vis(pr, "freq", .type = "boxplot")
vis(pr, "freq", .type = "none")
vis(pr, "freq", .type = "mean")
vis(pr, "freq", .by = "Status", .meta = immdata$meta)
```

vis_textlogo *Sequence logo plots for amino acid profiles.*

Description

Plot sequence logo plots for visualising of amino acid motif sequences / profiles.

'vis_textlogo' plots sequences in a text format - each letter has the same height. Useful when there are no big differences between occurences of amino acids in the motif.

'vis_seqlogo' is a traditional sequence logo plots. Useful when there are one or two amino acids with clear differences in their occurrences.

Usage

```
vis_textlogo(.data, .replace.zero.with.na = TRUE, .width = 0.1, ...)

vis_seqlogo(.data, .scheme = "chemistry", ...)
```

Arguments

`.data` Output from the `kmer.profile` function.

`.replace.zero.with.na`
if TRUE then replace all zeros with NAs, therefore letters with zero frequency wont appear at the plot.

`.width` Width for jitter, i.e., how much points will scatter around the verical line. Pass 0 (zero) to plot points on the straight vertical line for each position.

`...` Not used here.

`.scheme` Character. An argumentt passed to geom_logo specifying how to colour symbols.

Value

A ggplot2 object.

See Also

getKmers, kmer_profile

Examples

```
data(immdata)
kmers <- getKmers(immdata$data[[1]], 5)
ppm <- kmer_profile(kmers, "prob")
vis(ppm, .plot = "text")
vis(ppm, .plot = "seq")

d <- kmer_profile(c("CASLL", "CASSQ", "CASGL"))
vis_textlogo(d)
vis_seqlogo(d)
```

Index