Package 'tcR'

August 29, 2016

Type Package

Title Advanced Data Analysis of Immune Receptor Repertoires

Version 2.2.1.11

Date 2016-04-22

Author Vadim Nazarov

Maintainer Vadim Nazarov <vdm.nazarov@gmail.com>

Description Platform for the advanced analysis of T cell receptor and Immunoglobulin repertoires data and visualisation of the analysis results.

License Apache License 2.0

Depends R (>= 2.15.1), ggplot2 (>= 1.0.0), dplyr (>= 0.4.0), gridExtra (>= 0.9.0), reshape2 (>= 1.2.0), igraph (>= 0.7.1)

Imports utils (>= 3.1.0), Rcpp (>= 0.11.1), grid (>= 3.0.0), data.table (>= 1.9.0), gtable (>= 0.1.2), stringdist (>= 0.7.3), scales (>= 0.3.0)

Suggests knitr (>= 1.8), roxygen2 (>= 3.0.0)

LinkingTo Rcpp

URL http://imminfo.github.io/tcr/

BugReports https://github.com/imminfo/tcr/issues

VignetteBuilder knitr

RoxygenNote 5.0.1

NeedsCompilation yes

Repository CRAN

Date/Publication 2016-04-22 13:10:15

R topics documented:

AA_TABLE *Tables with genetic code.*

Description

Tables with genetic code.

Usage

```
AA_TABLE
```

Format

AA_TABLE:

```
Class 'table'  Named chr [1:65] "K" "N" "K" "N" ... ..- attr(*, "names")= chr [1:65] "AAA" "AAC" "AAG"
```

AA_TABLE_REVERSED:

```
List of 22 $ *: chr [1:3] "TAA" "TAG" "TGA" $ A: chr [1:4] "GCA" "GCC" "GCG" "GCT"
$ C: chr [1:2] "TGC" "TGT" $ D: chr [1:2] "GAC" "GAT" ...
```

Examples

```
## Not run:
AA_TABLE['ATG']  #  => "M"
AA_TABLE_REVERSED['K']  #  => list(K = c("AAA", "AAG"))

## End(Not run)
```

---

apply.symm *Apply function to every 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 = T)

apply.asymm(.datalist, .fun, ..., .diag = NA, .verbose = T)
```

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 T then output a progress bar.

Value

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

Examples

```
## Not run:
# equivalent to intersectClonesets(immdata, 'a0e')
apply.symm(immdata, intersectClonesets, .type = 'a0e')

## End(Not run)
```

assymetry *Normalised log assymetry.*

Description

Compute the value of the normalised log assymetry measure for the given data.frames of the counts of shared clones.

Usage

```
assymetry(.alpha, .beta = NULL, .by = "CDR3.nucleotide.sequence")
```

Arguments

.alpha	First mitcr data.frame or a list with data.frames.
.beta	Second mitcr data.frame or NULL if .alpha is a list.
.by	Which column use to merge. See "Details".

Details

Merge two data frames by the given column and compute value Sum(Log((Percentage for shared clone S from alpha) / (Percentage for shared clone S from beta))) / (# of shared clones).

Value

Value of the normalised log assymetry measure for the given data.frames.

barcodes.to.reads *Rearrange columns with counts for clonesets.*

Description

Replace Read.count with Umi.count, recompute Percentage and sort data.

Usage

```
barcodes.to.reads(.data)
```

Arguments

.data	Data frame with columns "Umi.count" and "Read.count".

Value

Data frame with new "Read.count" and "Percentage" columns.

beta.prob *List with assembling probabilities of beta chain TCRs.*

Description

beta.prob is a list with probabilities of TCR assembling taken from Murugan et al. Statistical inference of the ge
of T-cell receptors from sequence repertoires. It's a list with the following elements:

- P.V - matrix with 1 column and row names stands for V-beta segments. Each element is a probability of choosing corresponding V-beta segment.

- P.del.D1 - matrix 17x17 with probabilities of choosing D5-D3 deletions for TRBD1.

- P.del.D1 - matrix 17x17 with probabilities of choosing D5-D3 deletions for TRBD2.

- P.ins.len - matrix with first columns stands for number of insertions, second and third columns filled with probability values of choosing corresponding number of insertions in VD- and DJ-junctions correspondingly.

- P.ins.nucl - data frame with probability of choosing a nucleotide in the insertion on junctions with known previous nucleotide. First column with names of nucleotides, 2-5 columns are probabilities of choosing adjacent nucleotide in VD-junction, 6-9 columns are probabilities of choosing adjacent nucleotide in DJ-junction.

- P.del.J - matrix with the first column "P.del.V" with number of deletions, and other columns with names for V-segments and with probabilities of corresponding deletions.

- P.del.J - matrix with the first column "P.del.J" with number of deletions, and other columns with names for J-segments and with probabilities of corresponding deletions.

- P.J.D - matrix with two columns ("TRBD1" and "TRBD2") and 13 rows with row names stands for J-beta segments. Each element is a mutual probability of choosing J-D segments.

Format

beta.prob is a list of matrices and data frames.

Examples

```
## Not run:
# Generate 10 kmers with adjacent nucleotide probability.
generate.kmers.prob(rep.int(10, 10), .probs=beta.prob$P.ins.nucl[,c(1, 2:5)])

## End(Not run)
```

bootstrap.tcr *Bootstrap for data frames in package tcR.*

Description

Resample rows (i.e., clones) in the given data frame and apply the given function to them.

Usage

```
bootstrap.tcr(.data, .fun = entropy.seg, .n = 1000, .size = nrow(.data),
  .sim = c("uniform", "percentage"), .postfun = function(x) {     unlist(x)
  }, .verbose = T, .prop.col = "Read.proportion", ...)
```

Arguments

.data	Data frame.
.fun	Function applied to each sample.
.n	Number of iterations (i.e., size of a resulting distribution).
.size	Size of samples. For .sim == "uniform" stands for number of rows to take. For .sim == "percentage" stands for number of UMIs / read counts to take.
.sim	A character string indicating the type of simulation required. Possible values are "uniform" or "percentage". See "Details" for more details of type of simulation.
.postfun	Function applied to the resulting list: list of results from each processed sample.
.verbose	if T then show progress bar.
.prop.col	Column with proportions for each clonotype.
...	Further values passed to .fun.

Details

Argument .sim can take two possible values: "uniform" (for uniform distribution), when each row can be taken with equal probability, and "perccentage" when each row can be taken with probability equal to its "Read.proportion" column.

Value

Either result from .postfun or list of length .n with values of .fun.

Examples

```
## Not run:
# Apply entropy.seg function to samples of size 20000 from immdata$B data frame for 100 iterations.
bootstrap.tcr(immdata[[2]], .fun = entropy.seg, .n = 100, .size = 20000, .sim = 'uniform')

## End(Not run)
```

`check.distribution` *Check for adequaty of distrubution.*

Description

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

Usage

```
check.distribution(.data, .do.norm = NA, .laplace = 1, .na.val = 0,
  .warn.zero = F, .warn.sum = T)
```

Arguments

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

Value

Numeric vector.

`clonal.space.homeostasis`
Clonal space homeostasis.

Description

Compute clonal space homeostatsis - statistics of how many space occupied by clones with specific proportions.

Usage

```
clonal.space.homeostasis(.data, .clone.types = c(Rare = 1e-05, Small = 1e-04,
  Medium = 0.001, Large = 0.01, Hyperexpanded = 1),
  .prop.col = "Read.proportion")
```

Arguments

.data	Cloneset data frame or list with such data frames.
.clone.types	Named numeric vector.
.prop.col	Which column to use for counting proportions.

See Also

vis.clonal.space

Examples

```
## Not run:
data(twb)
# Compute summary space of clones, that occupy
# [0, .05) and [.05, 1] proportion.
clonal.space.homeostasis(twb, c(Low = .05, High = 1)))
#        Low (0 < X <= 0.05) High (0.05 < X <= 1)
# Subj.A           0.9421980           0.05780198
# Subj.B           0.9239454           0.07605463
# Subj.C           0.8279270           0.17207296
# Subj.D           1.0000000           0.00000000
# I.e., for Subj.D sum of all read proportions for clones
# which have read proportion between 0 and .05 is equal to 1.

## End(Not run)
```

---

cloneset.stats *MiTCR data frame basic statistics.*

Description

Compute basic statistics of TCR repertoires: number of clones, number of clonotypes, number of in-frame and out-of-frame sequences, summary of "Read.count", "Umi.count" and other.

Usage

```
cloneset.stats(.data, .head = 0)

repseq.stats(.data, .head = 0)
```

Arguments

.data	tcR data frames or a list with tcR data frames.
.head	How many top clones use to comput summary.

Value

if .data is a data frame, than numeric vector with statistics. If .data is a list with data frames, than matrix with statistics for each data frame.

Examples

```
## Not run:
# Compute basic statistics of list with data frames.
cloneset.stats(immdata)
repseq.stats(immdata)

## End(Not run)
```

---

codon.variants *Functions for working with aminoacid sequences.*

Description

codon.variants - get all codon variants for the given nucleotide sequence with known corresponding aminoacid sequence.

translated.nucl.variants - get number of nucleotide sequences which can be translated to the given aminoacid sequence.

reverse.translation - get all nucleotide sequences, which can be traslated to the given aminoacid sequence.

Usage

```
codon.variants(.aaseq, .nucseq = sapply(1:length(.aaseq),
               function (i) paste0(rep('XXX', times = nchar(.aaseq[i])),
               collapse = '')))

translated.nucl.sequences(.aaseq, .nucseq = sapply(1:length(.aaseq),
                     function (i) paste0(rep('XXX', times = nchar(.aaseq[i])),
                        collapse = '')))

reverse.translation(.aaseq, .nucseq = paste0(rep('XXX', times = nchar(.aaseq)),
                  collapse = ''))
```

Arguments

.aaseq	Amino acid sequence.
.nucseq	Nucleotide sequence with 'X' letter at non-fixed positions. Other positions will be fixed.

Value

List with all possible variants for every aminoacid in .aaseq, number of sequences or character vector of candidate sequences.

Examples

```
codon.variants('ACT')
translated.nucl.sequences(c('ACT', 'CASSLQ'))
reverse.translation('T')  # -> "ACA" "ACC" "ACG" "ACT"
reverse.translation('T', 'XXT')  # -> "ACT"
translated.nucl.sequences('ACT', 'XXXXXXXC')
codon.variants('ACT', 'XXXXXXXC')
reverse.translation('ACT', 'XXXXXXXC')
```

`column.summary` *Columns statistics.*

Description

`column.summary` - general function for computing summary statistics (using the `summary` function) for columns of the given mitcr data.frame: divide `.factor.column` by factors from `.alphabet` and compute statistics of correspondingly divided `.target.column`.

`insertion.stats` - compute statistics of insertions for the given mitcr data.frame.

Usage

```
column.summary(.data, .factor.col, .target.col, .alphabet = NA, .remove.neg = T)

insertion.stats(.data)
```

Arguments

`.data`	Data frame with columns `.factor.col` and `target.col`
`.factor.col`	Columns with factors by which the data will be divided to subsets.
`.target.col`	Column with numeric values for computing summaries after dividing the data to subsets.
`.alphabet`	Character vector of factor levels. If NA than use all possible elements frim the `.factor.col` column.
`.remove.neg`	Remove all elements which >-1 from the `.target.col` column.

Value

Data.frame with first column with levels of factors and 5 columns with output from the `summary` function.

See Also

summary

Examples

```
## Not run:
# Compute summary statistics of VD insertions
# for each V-segment using all V-segments in the given data frame.
column.summary(immdata[[1]], 'V.gene', 'Total.insertions')
# Compute summary statistics of VD insertions for each V-segment using only V-segments
# from the HUMAN_TRBV_MITCR
column.summary(immdata[[1]], 'V.gene', 'Total.insertions', HUMAN_TRBV_MITCR)

## End(Not run)
```

`contamination.stats` *Contamination filtering.*

Description

Occasionally DNA or RNA libraries are contaminate each other. To address this issue and estimate contamination rate `tcR` offers `contamination.stats` and `decontamination` functions. The `decontamination` function received data (either data frame or a list with data frames) and a limit for clonal proportion as arguments. Script searches for a similar clones to the first data frame in the other (or performs pairwise searches if the given data is a list) and removes clones from the first data frame, which has been found in the second one with counts less or equal to 10 * counts of similar clones in the first one. Function `contamination.stats` will return the number of clones which will be removed with the `contamination.stats` function.

Usage

```
contamination.stats(.data1, .data2, .limit = 20, .col = 'Read.count')

decontamination(.data1, .data2, .limit = 20, .col = 'Read.count', .symm = T)
```

Arguments

`.data1`	First data frame with columns 'CDR3.nucleotide.sequence' and 'Read.count'. Will be checked for contamination.
`.data2`	Second data frame with such columns. Will be used for checking for sequences which contaminated the first one.
`.limit`	Parameter for filtering: all sequences from `.data1` which are presented in `.data2` and (count of in `.data2`) / (count of seq in `.data1`) >= `.limit` are removed.
`.col`	Column's name with clonal count.
`.symm`	if T then perform filtering out of sequences in .data1, and then from .data2. Else only from .data1.

Value

Filtered `.data1` or a list with filtered both `.data1` and `.data2`.

convergence.index *Compute convergence characteristics of repertoires.*

Description

Get a number of rows with similar aminoacid sequence but different nucleotide sequence.

Usage

```
convergence.index(.alpha, .beta, .col.nuc = "CDR3.nucleotide.sequence",
  .col.aa = "CDR3.amino.acid.sequence")
```

Arguments

.alpha	Either data frame with columns .col.nuc and .col.aa or list with such data frames.
.beta	Either data frame or none.
.col.nuc	Name of the column with nucleotide sequences.
.col.aa	Name of the columnw ith aminoacid sequences.

Value

If .alpha is data frame, than integer vector of length 2 with . If .alpha is a list than matrix M with M[i,j] = convergence.index(.alpha[[i]], .alpha[[j]]).

cosine.sharing *Shared repertoire analysis.*

Description

Functions for computing statistics and analysis of shared repertoire of sequences.

cosine.sharing - apply the cosine similarity measure to the vectors of sequences' counts or indices.

shared.representation - for every repertoire in the shared repetoire get a number of sequences in this repertoire which are in the other repertoires. Row names of the input matrix is the number of people.

shared.clones.count - get the number of shared clones for every number of people.

shared.summary - get a matrix with counts of pairwise shared sequences (like a result from cross function, applied to a list of data frames).

Usage

```
cosine.sharing(.shared.rep, .log = T)

shared.representation(.shared.rep)

shared.clones.count(.shared.rep)

shared.summary(.shared.rep, .min.ppl = min(.shared.rep$People),
               .max.ppl = max(.shared.rep$People))
```

Arguments

.shared.rep	Shared repertoire, obtained from the function shared.repertoire.
.log	if T then apply log to the after adding laplace correction equal to one.
...	Parameters passed to the prcomp function.
.min.ppl	Filter: get sequences with # people >= .min.ppl.
.max.ppl	Filter: get sequences with # people <= .max.ppl.

Value

Plot or PCA resulr for the shared.seq.pca function or a matrix with cosine similarity values for the cosine.sharing function.

See Also

shared.repertoire

Examples

```
## Not run:
# Load the twb data.
data(twb)
# Create shared repertoire on the twins data using CDR3 amino acid sequences with CDR1-2.
twb.shared <- shared.repertoire(twb, 'av', .verbose = T)
sh.repr <- shared.representation(twb.shared)
sh.repr
# Get proportion of represented shared sequences.
apply(sh.repr, 2, function (col) col / col[1])

## End(Not run)
```

`cosine.similarity` *Set and vector similarity measures.*

Description

Functions for computing similarity between two vectors or sets. See "Details" for exact formulas.

- Cosine similarity is a measure of similarity between two vectors of an inner product space that measures the cosine of the angle between them.

- Tversky index is an asymmetric similarity measure on sets that compares a variant to a prototype.

- Overlap cofficient is a similarity measure related to the Jaccard index that measures the overlap between two sets, and is defined as the size of the intersection divided by the smaller of the size of the two sets.

- Jaccard index is a statistic used for comparing the similarity and diversity of sample sets.

- Morisita's overlap index is a statistical measure of dispersion of individuals in a population. It is used to compare overlap among samples (Morisita 1959). This formula is based on the assumption that increasing the size of the samples will increase the diversity because it will include different habitats (i.e. different faunas).

- Horn's overlap index based on Shannon's entropy.

Use the repOverlap function for computing similarities of clonesets.

Usage

```
cosine.similarity(.alpha, .beta, .do.norm = NA, .laplace = 0)

tversky.index(x, y, .a = 0.5, .b = 0.5)

overlap.coef(.alpha, .beta)

jaccard.index(.alpha, .beta, .intersection.number = NA)

morisitas.index(.alpha, .beta, .do.unique = T)

horn.index(.alpha, .beta, .do.unique = T)
```

Arguments

`.alpha, .beta, x, y`	Vector of numeric values for cosine similarity, vector of any values (like characters) for `tversky.index` and `overlap.coef`, matrix or data.frame with 2 columns for `morisitas.index` and `horn.index`, either two sets or two numbers of elements in sets for `jaccard.index`.
`.do.norm`	One of the three values - NA, T or F. If NA than check for distrubution (sum(.data) == 1) and normalise if needed with the given laplace correction value. if T then do normalisation and laplace correction. If F than don't do normalisaton and laplace correction.

`.laplace`	Value for Laplace correction.
`.a, .b`	Alpha and beta parameters for Tversky Index. Default values gives the Jaccard index measure.
`.do.unique`	if T then call unique on the first columns of the given data.frame or matrix.
`.intersection.number`	Number of intersected elements between two sets. See "Details" for more information.

Details

For `morisitas.index` input data are matrices or data.frames with two columns: first column is elements (species or individuals), second is a number of elements (species or individuals) in a population.

Formulas:

Cosine similarity: `cos(a, b) = a * b / (||a|| * ||b||)`

Tversky index: `S(X, Y) = |X and Y| / (|X and Y| + a*|X - Y| + b*|Y - X|)`

Overlap coefficient: `overlap(X, Y) = |X and Y| / min(|X|, |Y|)`

Jaccard index: `J(A, B) = |A and B| / |A U B|` For Jaccard index user can provide |A and B| in `.intersection.number` otherwise it will be computed using `base::intersect` function. In this case `.alpha` and `.beta` expected to be vectors of elements. If `.intersection.number` is provided than `.alpha` and `.beta` are exptected to be numbers of elements.

Formula for Morisita's overlap index is quite complicated and can't be easily shown here, so just look at this webpage: http://en.wikipedia.org/wiki/Morisita

Value

Value of similarity between the given sets or vectors.

See Also

repOverlap, intersectClonesets, entropy, diversity

Examples

```
## Not run:
jaccard.index(1:10, 2:20)
a <- length(unique(immdata[[1]][, c('CDR3.amino.acid.sequence', 'V.gene')]))
b <- length(unique(immdata[[2]][, c('CDR3.amino.acid.sequence', 'V.gene')]))
# Next
jaccard.index(a, b, repOverlap(immdata[1:2], .seq = 'aa', .vgene = T))
# is equal to
repOverlap(immdata[1:2], 'jaccard', seq = 'aa', .vgene = T)

## End(Not run)
```

`entropy` *Information measures.*

Description

Functions for information measures of and between distributions of values.

Warning! Functions will check if `.data` if a distribution of 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). For `js.div` and `kl.div` vectors of values must have equal length.

Functions:

- The Shannon entropy quantifies the uncertainty (entropy or degree of surprise) associated with this prediction.

- Kullback-Leibler divergence (information gain, information divergence, relative entropy, KLIC) is a non-symmetric measure of the difference between two probability distributions P and Q (measure of information lost when Q is used to approximate P).

- Jensen-Shannon divergence is a symmetric version of KLIC. Square root of this is a metric often referred to as Jensen-Shannon distance.

Usage

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

kl.div(.alpha, .beta, .do.norm = NA, .laplace = 1e-12)

js.div(.alpha, .beta, .do.norm = NA, .laplace = 1e-12, .norm.entropy = F)
```

Arguments

`.data, .alpha, .beta`	Vector of values.
`.norm`	if T then compute normalised entropy (H / Hmax).
`.do.norm`	One of the three values - NA, T or F. If NA than check for distrubution `(sum(.data) == 1)`. and normalise if needed with the given laplace correction value. if T then do normalisation and laplace correction. If F than don't do normalisaton and laplace correction.
`.laplace`	Value for Laplace correction which will be added to every value in the .data.
`.norm.entropy`	if T then normalise JS-divergence by entropy.

Value

Shannon entropy, Jensen-Shannon divergence or Kullback-Leibler divergence values.

See Also

similarity, diversity

`entropy.seg` *Repertoires' analysis using information measures applied to V- and J-segment frequencies.*

Description

Information approach to repertoire analysis. Function `entropy.seg` applies Shannon entropy to V-usage and hence measures variability of V-usage. Function `js.div.seg` applied Jensen-Shannon divergence to V-usage of two or more data frames and hence measures distance among this V-usages.

Usage

```
entropy.seg(.data, .genes = HUMAN_TRBV, .frame = c('all', 'in', 'out'),
            .quant = c(NA, "read.count", "umi.count", "read.prop", "umi.prop"),
            .ambig = F)

js.div.seg(.data, .genes = HUMAN_TRBV, .frame = c('all', 'in', 'out'),
           .quant = c(NA, "read.count", "umi.count", "read.prop", "umi.prop"),
           .norm.entropy = T, .ambig = F, .verbose = F, .data2 = NULL)
```

Arguments

`.data`	Mitcr data.frame or a list with mitcr data.frames.
`.genes`	Parameter to the geneUsage function.
`.frame`	Character vector of length 1 specified which *-frames should be used: only in-frame ('in'), out-of-frame ('out') or all sequences ('all').
`.quant`	Which column to use for the quantity of clonotypes: NA for computing only number of genes without using clonotype counts, "read.count" for the "Read.count" column, "umi.count" for the "Umi.count" column, "read.prop" for the "Read.proportion" column, "umi.prop" for the "Umi.proportion" column.
`.ambig`	Parameter passed to geneUsage.
`.data2`	NULL if .data is a list, or a second mitcr data.frame.
`.norm.entropy`	if T then divide result by mean entropy of 2 segments' frequencies.
`.verbose`	If T than output the data processing progress bar.

Value

For `entropy.seg` - numeric integer with entropy value(s). For `js.div.seg` - integer of vector one if `.data` and `.data2` are provided; esle matrix length(.data) X length(.data) if `.data` is a list.

See Also

vis.heatmap, vis.group.boxplot, geneUsage

`find.clonotypes` *Find target clonotypes and get columns' value corresponded to that clonotypes.*

Description

Find the given target clonotypes in the given list of data.frames and get corresponding values of desired columns.

Usage

```
find.clonotypes(.data, .targets, .method = c("exact", "hamm", "lev"),
  .col.name = "Read.count", .target.col = "CDR3.amino.acid.sequence",
  .verbose = T)
```

Arguments

`.data`	List with mitcr data.frames or a mitcr data.frame.
`.targets`	Target sequences or elements to search. Either character vector or a matrix / data frame (not a data table!) with two columns: first for sequences, second for V-segments.
`.method`	Method, which will be used to find clonotypes: - "exact" performs exact matching of targets; - "hamm" finds targets and close sequences using hamming distance <= 1; - "lev" finds targets and close sequences using levenshtein distance <= 1.
`.col.name`	Character vector with column names which values should be returned.
`.target.col`	Character vector specifying name of columns in which function will search for a targets. Only first column's name will be used for matching by different method, others will match exactly. `.targets` should be a two-column character matrix or data frame with second column for V-segments.
`.verbose`	if T then print messages about the search process.

Value

Data.frame.

Examples

```
## Not run:
# Get ranks of all given sequences in a list of data frames.
immdata <- set.rank(immdata)
find.clonotypes(.data = immdata, .targets = head(immdata[[1]]$CDR3.amino.acid.sequence),
             .method = 'exact', .col.name = "Rank", .target.col = "CDR3.amino.acid.sequence")
# Find close by levenhstein distance clonotypes with similar V-segments and return
# their values in columns 'Read.count' and 'Total.insertions'.
find.clonotypes(.data = twb, .targets = twb[[1]][, c('CDR3.amino.acid.sequence', 'V.gene')],
```

```
                .col.name = c('Read.count', 'Total.insertions'), .method = 'lev',
                .target.col = c('CDR3.amino.acid.sequence', 'V.gene'))

## End(Not run)
```

`find.similar.sequences` *Find similar sequences.*

Description

Return matrix M with two columns. For each element in row i and column j M[i,j] => distance between pattern(i) and data(j) sequences equal to or less than .max.errors. This function will uppercase .data and remove all strings, which have anything than A-Z letters.

Usage

```
find.similar.sequences(.data, .patterns = c(), .method = c('exact', 'hamm', 'lev'),
                       .max.errors = 1, .verbose = T, .clear = F)

exact.match(.data, .patterns = c(), .verbose = T)

hamming.match(.data, .patterns = c(), .max.errors = 1, .verbose = T)

levenshtein.match(.data, .patterns = c(), .max.errors = 1, .verbose = T)
```

Arguments

`.data`	Vector of strings.
`.patterns`	Character vector of sequences, which will be used for searching for neighbours.
`.method`	Which method use: 'exact' for exact matching, 'hamm' for Hamming Distance, 'lev' for Levenshtein distance.
`.max.errors`	Max Hamming or Levenshtein distance between strings. Doesn't use in 'exact' setting.
`.verbose`	Should function print progress or not. // DON'T USE IT
`.clear`	if T then remove all sequences with character "*" or "~".

Value

Matrix with two columns [i,j], dist(data(i), data(j)) <= .max.errors.

`fix.alleles` *Fix alleles / genes by removing allele information / unnecessary colons.*

Description

Fix alleles / genes by removing allele information / unnecessary colons.

Usage

```
fix.alleles(.data)
```

Arguments

`.data`	tcR data frame.

`gc.content` *GC-content of a nucleotide sequences.*

Description

Compute the GC-content (proportion of G-C nucleotide in a sequence).

Usage

```
gc.content(.nucseq)
```

Arguments

`.nucseq`	Character vector of nucletoide sequences.

Value

Numeric vector of `length(.nucseq)`.

generate.kmers *Generate k-mers.*

Description

Generate all k-mers. starting with the given sequence on the given alphabet Generate k-mers with the given k and probabilities of nucleotides next to each other (markov chain).

Usage

```
generate.kmers(.k, .seq = '', .alphabet = c('A', 'C', 'G', 'T'))

generate.kmers.prob(.k, .probs, .count = 1, .alphabet = c('A', 'C', 'G', 'T'),
                    .last.nucl = 'X')
```

Arguments

.k	Size of k-mers or either integer or vector with k-s for generate.kmers.prob.
.seq	Prefix of all generated k-mers.
.alphabet	Alphabet.
.probs	Matrix with probabilities for generating adjacent symbol with |alphabet| X |alphabet| size. Order of letters is given in the .alphabet parameter.
.count	Number of kmers to be generated.
.last.nucl	Adjacent nucleotide from which start generation. If 'X' than choose one of the nucleotides with equal probabilities.

Value

Vector of all possible k-mers for generate.kmers or a vector of generated kmers for generate.kmers.prob.

generate.tcr *Generate random nucleotide TCR sequences.*

Description

Given the list of probabilities and list of segments (see "Details"), generate a artificial TCR repertoire.

Usage

```
generate.tcr(.count = 1, .chain = c("beta", "alpha"), .segments,
  .P.list = if (.chain[1] == "alpha") alpha.prob else beta.prob)
```

Arguments

.count	Number of TCR sequences to generate.
.chain	Either "alpha" or "beta" for alpha and beta chain respectively.
.segments	List of segments (see "Details").
.P.list	List of probabilities (see "Details").

Details

For the generation of a artifical TCR repertoire user need to provide two objects: the list with segments and the list with probabilities. List with segments is a list of 5 elements with 5 names: "TRAV", "TRAJ", "TRBV", "TRBD", "TRBJ". Each element is a data frame with following columns (order is matters!): "V.allelles" with names for V-segments (for TRAV and TRBV; for others is "J.allelles" or "D.allelles"), "CDR3.position" (the function doesn't use it, but you should provide it, fill it with zeros, for example), "Full.nucleotide.sequence" (the function doesn't use it), "Nucleotide.sequence" (function uses it for getting nucleotide sequences of segments) and "Nucleotide.sequence.P" (the function doesn't use it).

List with probabilities is quite complicated, so just call `data(beta.prob)` for beta chain probabilities (alpha chain probabilities will be added soon).

Value

Mitcr data.frame with generated sequences.

See Also

genesegments beta.prob

Examples

```
## Not run:
# Load list of segments provided along with tcR.
data(genesegments)
# Load list of probabilities provided along with tcR.
data(beta.prob)
# Generate repertoire of beta chian with 10000 sequences.
artif.rep <- generate.tcR(10000, 'beta')
View(artif.rep)

## End(Not run)
```

geneUsage *Gene usage.*

Description

Compute frequencies or counts of gene segments ("V / J - usage").

Usage

```
geneUsage(.data, .genes = HUMAN_TRBV_MITCR, .quant = c(NA, "read.count",
  "umi.count", "read.prop", "umi.prop"), .norm = F, .ambig = F)
```

Arguments

.data	Cloneset data frame or a list with clonesets.
.genes	Either one of the gene alphabet (e.g., HUMAN_TRBV, genealphabets) or list with two gene alphabets for computing joint distribution.
.quant	Which column to use for the quantity of clonotypes: NA for computing only number of genes without using clonotype counts, "read.count" for the "Read.count" column, "umi.count" for the "Umi.count" column, "read.prop" for the "Read.proportion" column, "umi.prop" for the "Umi.proportion" column.
.norm	If T then return proportions of resulting counting of genes.
.ambig	If F than remove from counting genes which are not presented in the given gene alphabet(s).

Value

If .data is a cloneset and .genes is NOT a list than return a data frame with first column "Gene" with genes and second with counts / proportions.

If .data is a list with clonesets and .genes is NOT a list than return a data frame with first column "Gene" with genes and other columns with counts / proportions for each cloneset in the input list.

If .data is a cloneset and .genes IS a list than return a matrix with gene segments for the first gene in .genes and column names for the second gene in .genes. See "Examples".

If .data is a list with clonesets and .genes IS a list than return a list with matrices like in the previous case.

See Also

genealphabets, vis.gene.usage, pca.segments

Examples

```
## Not run:
# Load your data
data(twb)
# compute V-segments frequencies of human TCR beta.
seg <- geneUsage(twb, HUMAN_TRBV, .norm = T)
# plot V-segments frequencies as a heatmap
vis.heatmap(seg, .labs = c("Sample", "V gene"))
# plot V-segments frequencies directly from clonesets
vis.gene.usage(twb, HUMAN_TRBV)
# plot V-segments frequencies from the gene frequencies
vis.gene.usage(seg, NA)
# Compute V-J joint usage.
geneUsage(twb, list(HUMAN_TRBV, HUMAN_TRBJ))
# for future:
# geneUsage(twb, "human", "trbv")

## End(Not run)
```

---

get.all.substrings *Get all substrings for the given sequence.*

Description

Get all substrings for the given sequence.

Usage

```
get.all.substrings(.seq, .min.len = 3, .table = T)
```

Arguments

.seq	Sequence for splitting to substrings.
.min.len	Minimal length of output sequences.
.table	if T then return data frame with substrings and positions of their ends in the .seq.

Value

Character vector or data frame with columns "Substring", "Start" and "End".

`get.deletions.alpha` *Compute the number of deletions in MiTCR data frames.*

Description

Get deletions for VD, DJ, 5'D and 3'D ends and two columns with total deletions for VD/DJ and 5'D/3'D deletions for the given mitcr data.frame with 0-indexes columns. Cases, in which deletions cannot be determined, will have -1 in their cell.

Usage

```
get.deletions.alpha(.data, .Vs = segments$TRAV, .Js = segments$TRAJ)

get.deletions.beta(.data, .Vs = segments$TRBV, .Js = segments$TRBJ, .Ds = segments$TRBD)
```

Arguments

`.data`	Mitcr data.frame.
`.Vs`	Table of V segments; must have 'V.segment' and 'Nucleotide.sequence' columns.
`.Js`	Table of J segments; must have 'J.segment' and 'Nucleotide.sequence' columns.
`.Ds`	Table of D segments; must have 'D.segment' and 'Nucleotide.sequence' columns.

Details

By default, `*.table` parameters are taken from the `segments` data frame which can be loaded to your R environment with data(segments). Data for segments has been taken from IMGT.

Value

Mitcr data.frame with 3 (for alpha chains) or 5 (for beta chains) new columns for deletions.

Examples

```
## Not run:
data(segments)
immdata <- get.deletions.beta(.data)
immdata.prob <- tcr.prob.df(immdata)

## End(Not run)
```

get.inframes *In-frame / out-of-frame sequences filter.*

Description

Return the given data frame with in-frame or out-of-frame sequences only. Nucleotide sequences in a column "CDR3.nucleotide.sequence" are checked if they length are divisible by 3 (len mod 3 == 0 => in-frame, else out-of-frame)

Usage

```
get.inframes(.data, .head = 0, .coding = T)

get.outframes(.data, .head = 0)

count.inframes(.data, .head = 0, .coding = T)

count.outframes(.data, .head = 0)

get.frames(.data, .frame = c('in', 'out', 'all'), .head = 0, .coding = T)

count.frames(.data, .frame = c('in', 'out', 'all'), .head = 0, .coding = T)
```

Arguments

.data	MiTCR data.frame or a list with mitcr data.frames.
.head	Parameter to the head() function. Supply 0 to get all elements. head applied before subsetting, i.e. if .head == 500, you will get in-frames from the top 500 clonotypes.
.coding	if T then return only coding sequences, i.e. without stop-codon.
.frame	Which *-frames to choose.

Value

Filtered data.frame or a list with such data.frames.

get.kmers *Get kmers from sequences.*

Description

Get vector of kmers from the given character vector or data frame.

Usage

```
get.kmers(.data, .head = -1, .k = 5, .clean = T, .meat = F,
  .verbose = T, .left.shift = 0, .right.shift = 0)
```

Arguments

.data	Either character vector or a data.frame.
.head	Parameter for head function applied to the given data before kmer generation.
.k	Size of the kmer.
.clean	if T then remove sequences which contain '~' or '*' symbols. Useful for deleting out-of-frame aminoacid sequnces.
.meat	if TRUE than .data must be data.frame with columns CDR3.amino.acid.sequence and Read.count.
.verbose	if T then print progress.
.left.shift	Cut all .left.shift symbols from the left side for each sequence.
.right.shift	Cut all .right.shift symbols from the right side for each sequence.

Value

Data.frame with 2 columns Kmers and Count / Rank / Proportion relatively to the .value param or a list with such data.frames if .data is a list.

---

gibbs.sampler *Gibbs Sampler.*

Description

Perform the Gibbs Sampler method for finding frequent motifs in the given vector of strings or data.frame. Each string splitted to kmers with the given length of motif.

Usage

```
gibbs.sampler(.data, .k = 5, .niter = 500)
```

Arguments

.data	Vector of characters or data.frame of characters (1st col) and their numbers (2nd col) if .meat == T.
.k	Motif's length.
.niter	Number of iterations.

Value

Vector of possible motifs.

`group.clonotypes` *Get all unique clonotypes.*

Description

Get all unique clonotypes with merged counts. Unique clonotypes are those with either equal CDR3 sequence or with equal CDR3 sequence and equal gene segments. Counts of equal clonotypes will be summed up.

Usage

```
group.clonotypes(.data, .gene.col = "V.gene", .count.col = "Read.count",
  .prop.col = "Read.proportion", .seq.col = "CDR3.amino.acid.sequence")
```

Arguments

`.data`	Either tcR data frame or a list with data frames.
`.gene.col`	Either name of the column with gene segments used to compare clonotypes or NA if you don't need comparing using gene segments.
`.count.col`	Name of the column with counts for each clonotype.
`.prop.col`	Name of the column with proportions for each clonotype.
`.seq.col`	Name of the column with clonotypes' CDR3 sequences.

Value

Data frame or a list with data frames with updated counts and proportion columns and rows with unique clonotypes only.

Examples

```
## Not run:
tmp <- data.frame(A = c('a','a','b','c', 'a')
B = c('V1', 'V1','V1','V2', 'V3')
C = c(10,20,30,40,50), stringsAsFactors = F)
tmp
#   A  B  C
# 1 a V1 10
# 2 a V1 20
# 3 b V1 30
# 4 c V2 40
# 5 a V3 50
group.clonotypes(tmp, 'B', 'C', 'A')
#  A  B  C
#  1 a V1 30
#  3 b V1 50
#  4 c V2 30
#  5 a V3 40
group.clonotypes(tmp, NA, 'C', 'A')
```

```
#   A  B  C
# 1 a V1 80
# 3 b V1 30
# 4 c V2 40
# For tcR data frame:
data(twb)
twb1.gr <- group.clonotypes(twb[[1]])
twb.gr <- group.clonotypes(twb)

## End(Not run)
```

`has.class` *Check if a given object has a given class.*

Description

Check if a given object has a given class.

Usage

```
has.class(.data, .class)
```

Arguments

`.data`	Object.
`.class`	String naming a class.

Value

Logical.

`intersectClonesets` *Intersection between sets of sequences or any elements.*

Description

Functions for the intersection of data frames with TCR / Ig data. See the `repOverlap` function for a general interface to all overlap analysis functions.

`intersectClonesets` - returns number of similar elements in the given two clonesets / data frames or matrix with counts of similar elements among each pair of objects in the given list.

`intersectCount` - similar to `tcR::intersectClonesets`, but with fewer parameters and only for two objects.

`intersectIndices` - returns matrix M with two columns, where element with index M[i, 1] in the first given object is similar to an element with index M[i, 2] in the second given object.

`intersectLogic` - returns logic vector with TRUE values in positions, where element in the first given data frame is found in the second given data frame.

Usage

```
intersectClonesets(.alpha = NULL, .beta = NULL, .type = "n0e", .head = -1, .norm = F,
          .verbose = F)

intersectCount(.alpha, .beta, .method = c('exact', 'hamm', 'lev'), .col = NULL)

intersectIndices(.alpha, .beta, .method = c('exact', 'hamm', 'lev'), .col = NULL)

intersectLogic(.alpha, .beta, .method = c('exact', 'hamm', 'lev'), .col = NULL)
```

Arguments

.alpha	Either first vector or data.frame or list with data.frames.
.beta	Second vector or data.frame or type of intersection procedure (see the .type parameter) if .alpha is a list.
.type	Types of intersection procedure if .alpha and .beta is data frames. String with 3 characters (see 'Details' for more information).
.head	Parameter for the head function, applied before intersecting.
.norm	If TRUE than normalise result by product of length or nrows of the given data.
.verbose	if T then produce output of processing the data.
.method	Method to use for intersecting string elements: 'exact' for exact matching, 'hamm' for matching strings which have <= 1 hamming distance, 'lev' for matching strings which have <= 1 levenshtein (edit) distance between them.
.col	Which columns use for fetching values to intersect. First supplied column matched with .method, others as exact values.

Details

Parameter .type of the intersectClonesets function is a string of length 3 [0an][0vja][ehl], where:

1. First character defines which elements intersect ("a" for elements from the column "CDR3.amino.acid.sequence", "n" for elements from the column "CDR3.nucleotide.sequence", other characters - intersect elements as specified);
2. Second character defines which columns additionaly script should use ('0' for cross with no additional columns, 'v' for cross using the "V.gene" column, 'j' for cross using "J.gene" column, 'a' for cross using both "V.gene" and "J.gene" columns);
3. Third character defines a method of search for similar sequences is use: "e" stands for the exact match of sequnces, "h" for match elements which have the Hamming distance between them equal to or less than 1, "l" for match elements which have the Levenshtein distance between tham equal to or less than 1.

Value

intersectClonesets returns (normalised) number of similar elements or matrix with numbers of elements.

`intersectCount` returns number of similar elements.

`intersectIndices` returns 2-row matrix with the first column stands for an index of an element in the given `x`, and the second column stands for an index of an element of `y` which is similar to a relative element in `x`;

`intersectLogic` returns logical vector of `length(x)` or `nrow(x)`, where TRUE at position `i` means that element with index `i` has been found in the `y`

See Also

repOverlap, vis.heatmap, ozScore, permutDistTest, vis.group.boxplot

Examples

```
## Not run:
data(twb)
# Equivalent to intersectClonesets(twb[[1]]$CDR3.nucleotide.sequence,
#                          twb[[2]]$CDR3.nucleotide.sequence)
# or intersectCount(twb[[1]]$CDR3.nucleotide.sequence,
#                   twb[[2]]$CDR3.nucleotide.sequence)
# First "n" stands for a "CDR3.nucleotide.sequence" column, "e" for exact match.
twb.12.n0e <- intersectClonesets(twb[[1]], twb[[2]], 'n0e')
stopifnot(twb.12.n0e == 46)
# First "a" stands for "CDR3.amino.acid.sequence" column.
# Second "v" means that intersect should also use the "V.gene" column.
intersectClonesets(twb[[1]], twb[[2]], 'ave')
# Works also on lists, performs all possible pairwise intersections.
intersectClonesets(twb, 'ave')
# Plot results.
vis.heatmap(intersectClonesets(twb, 'ave'), .title = 'twb - (ave)-intersection', .labs = '')
# Get elements which are in both twb[[1]] and twb[[2]].
# Elements are tuples of CDR3 nucleotide sequence and corresponding V-segment
imm.1.2 <- intersectLogic(twb[[1]], twb[[2]],
                          .col = c('CDR3.amino.acid.sequence', 'V.gene'))
head(twb[[1]][imm.1.2, c('CDR3.amino.acid.sequence', 'V.gene')])
data(twb)
ov <- repOverlap(twb)
sb <- matrixSubgroups(ov, list(tw1 = c('Subj.A', 'Subj.B'), tw2 = c('Subj.C', 'Subj.D')));
vis.group.boxplot(sb)

## End(Not run)
```

`inverse.simpson` *Distribution evaluation.*

Description

Functions for evaluating the diversity of species or objects in the given distribution. See the `repOverlap` function for working with clonesets and a general interface to all of this functions.

Warning! 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.

- 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).

Usage

```
inverse.simpson(.data, .do.norm = NA, .laplace = 0)

diversity(.data, .q = 5, .do.norm = NA, .laplace = 0)

gini(.data, .do.norm = NA, .laplace = 0)

gini.simpson(.data, .do.norm = NA, .laplace = 0)

chao1(.data)
```

Arguments

.data	Numeric vector of values for proportions or for numbers of individuals.
.do.norm	One of the three values - NA, T or F. If NA than check for distrubution (sum(.data) == 1) and normalise if needed with the given laplace correction value. if T then do normalisation and laplace correction. If F than don't do normalisaton and laplace correction.
.laplace	Value for Laplace correction which will be added to every value in the .data.
.q	q-parameter for the Diversity index.

Value

Numeric vector of length 1 with value for all functions except chao1, which returns 4 values: estimated number of species, standart deviation of this number and two 95

See Also

repOverlap, entropy, similarity

Examples

```
data(twb)
# Next two are equal calls:
stopifnot(gini(twb[[1]]$Read.count, TRUE, 0) - 0.7609971 < 1e-07)
stopifnot(gini(twb[[1]]$Read.proportion, FALSE) - 0.7609971 < 1e-07)
stopifnot(chao1(twb[[1]]$Read.count)[1] == 1e+04)
```

---

kmer.profile *Profile of sequences of equal length.*

Description

Return profile for the given character vector or a data frame with sequences of equal length or list with them.

Usage

```
kmer.profile(.data, .names = rep('Noname', times=length(.data)), .verbose = F)
```

Arguments

.data	Either list with elements of one of the allowed classes or object with one of the class: data.frame with first column with character sequences and second column with number of sequences or character vector.
.names	Names for each sequence / row in the .data.
.verbose	if T then print processing output.

Value

Return data frame with first column "Symbol" with all possible symbols in the given sequences and other columns with names "1", "2", ... for each position with percentage for each symbol.

See Also

vis.logo

kmer.table *Make and manage the table of the most frequent k-mers.*

Description

kmer.table - generate table with the most frequent k-mers.

get.kmer.column - get vector of k-mers from the k-mer table from the function kmer.table

Usage

```
kmer.table(.data, .heads = c(10, 100, 300, 1000, 3000, 10000, 30000), .nrow = 20,
           .clean = T, .meat = F)

get.kmer.column(.kmer.table.list, .head)
```

Arguments

.data	Mitcr data.frame or a list with mitcr data.frames.
.heads	Vector of parameter for the head() function, supplied sequentialy to the get.kmers() function. -1 means all rows.
.nrow	How many most frequent k-mers include to the output table.
.clean	Parameter for the get.kmers() function.
.meat	Parameter for the get.kmers() function.
.kmer.table.list	Result from the kmer.table function if .data supplied as a list.
.head	Which columns with this head return.

Value

kmer.table - if .data is a data frame, than data frame with columns like "Kmers.X", "Count.X" where X - element from .heads. If .data is a list, than list of such data frames.

get.kmer.column - data frame with first column with kmers and other columns named as a names of data frames, from which .kmer.table.list was generated.

Examples

```
## Not run:
twb.kmers <- kmer.table(twb, .heads = c(5000, 10000), .meat = T)
head(get.kmer.column(twb.kmers, 10000))

## End(Not run)
```

loglikelihood *Log-likelihood.*

Description

Compute the log-likelihood of the given distribution or vector of counts.

Usage

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

Arguments

.data	Vector for distribution or counts.
.base	Logarightm's base for the loglikelihood.
.do.norm	Parameter to the check.distribution function.
.laplace	Laplace correction, Parameter to the check.distribution function.

Value

Loglikelihood value.

matrixdiagcopy *Copy the up-triangle matrix values to low-triangle.*

Description

Copy the up-triangle matrix values to low-triangle.

Usage

```
matrixdiagcopy(mat)
```

Arguments

mat	Given up-triangle matrix.

Value

Full matrix.

matrixSubgroups *Get all values from the matrix corresponding to specific groups.*

Description

Split all matrix values to groups and return them as a data frame with two columns: for values and for group names.

Usage

```
matrixSubgroups(.mat, .groups = NA, .symm = T, .diag = F)
```

Arguments

.mat	Input matrix with row and columns names.
.groups	Named list with character vectors for names of elements for each group.
.symm	If T than remove symmetrical values from the input matrix.
.diag	If .symm if T and .diag is F than remove diagonal values.

See Also

repOverlap, vis.group.boxplot

Examples

```
## Not run:
data(twb)
ov <- repOverlap(twb)
sb <- matrixSubgroups(ov, list(tw1 = c('Subj.A', 'Subj.B'), tw2 = c('Subj.C', 'Subj.D')));
vis.group.boxplot(sb)

## End(Not run)
```

mutated.neighbours *Get vertex neighbours.*

Description

Get all properties of neighbour vertices in a mutation network of specific vertices.

Usage

```
mutated.neighbours(.G, .V, .order = 1)
```

Arguments

.G	Mutation network.
.V	Indices of vertices for which return neighbours.
.order	Neighbours of which order return.

Value

List of length .V with data frames with vertex properties. First row in each data frame is the vertex for which neighbours was returned.

Examples

```
## Not run:
data(twb)
twb.shared <- shared.repertoire(twb)
G <- mutation.network(twb.shared)
head(mutated.neighbours(G, 1)[[1]])
#           label             vseg repind prob people npeople
# 1 CASSDRDTGELFF          TRBV6-4      1   -1   1111       4
# 2 CASSDSDTGELFF          TRBV6-4     69   -1   1100       2
# 3 CASSYRDTGELFF TRBV6-3, TRBV6-2    315   -1   1001       2
# 4 CASKDRDTGELFF TRBV6-3, TRBV6-2   2584   -1   0100       1
# 5 CASSDGDTGELFF          TRBV6-4   5653   -1   0010       1
# 6 CASSDRETGELFF          TRBV6-4   5950   -1   0100       1

## End(Not run)
```

---

mutation.network *Make mutation network for the given repertoire.*

Description

Mutation network (or a mutation graph) is a graph with vertices representing nucleotide or in-frame amino acid sequences (out-of-frame amino acid sequences will automatically filtered out) and edges are connecting pairs of sequences with hamming distance or edit distance between them no more than specified in the .max.errors function parameter.

Usage

```
mutation.network(.data, .method = c("hamm", "lev"), .max.errors = 1,
  .label.col = "CDR3.amino.acid.sequence", .seg.col = "V.gene",
  .prob.col = "Probability")
```

Arguments

.data	Either character vector of sequences, data frame with .label.col or shared repertoire (result from the shared.repertoire function) constructed based on .label.col.
.method	Either "hamm" (for hamming distance) or "lev" (for edit distance). Passed to the find.similar.sequences function.
.max.errors	Passed to the find.similar.sequences function.
.label.col	Name of the column with CDR3 sequences (vertex labels).
.seg.col	Name of the column with V gene segments.
.prob.col	Name of the column with clonotype probability.

Value

Mutation network, i.e. igraph object with input sequences as vertices labels, ???

See Also

shared.repertoire, find.similar.sequences, set.people.vector, get.people.names

Examples

```
## Not run:
data(twb)
twb.shared <- shared.repertoire(twb)
G <- mutation.network(twb.shared)
get.people.names(G, 300, T)  # "Subj.A|Subj.B"
get.people.names(G, 300, F)  # list(c("Subj.A", "Subj.B"))

## End(Not run)
```

---

ozScore *Overlap Z-score.*

Description

Compute OZ-scores ("overlap Z scores") for values in the given matrix of overlaps, i.e.,. for each value compute the number of standart deviations from the mean of the matrix.

Usage

```
ozScore(.mat, .symm = T, .as.matrix = F, .val.col = c("norm", "abs",
  "oz"))
```

Arguments

.mat	Matrix with overlap values.
.symm	If T then remove lower triangle matrix from counting. Doesn't work if the matrix has different number of rows and columns.
.as.matrix	If T then return
.val.col	If .as.matrix is T then this is a name of the column to build matrix upon: either "oz" for the OZ-score column, "abs" for the absolute OZ-score column, or "norm" for the normalised absolute OZ-score column.

See Also

repOverlap, intersectClonesets, permutDistTest

Examples

```
## Not run:
data(twb)
mat <- repOverlap(twb)
ozScore(mat)
# Take 3x3 matrix
ozScore(mat[1:3, 1:3])
# Return as matrix with OZ scores
ozmat <- ozScore(mat, T, T, 'oz')
# Return as matrix with normalised absolute OZ scores
oznorm <- ozScore(mat, T, T, 'norm')
# Plot it as boxplots
sb <- matrixSubgroups(oznorm, list(tw1 = c('Subj.A', 'Subj.B'), tw2 = c('Subj.C', 'Subj.D')));
vis.group.boxplot(sb)

## End(Not run)
```

---

parse.cloneset *Parse input table files with the immune receptor repertoire data.*

Description

General parser for cloneset table files. Each column name has specific purpose (e.g., column for CDR3 nucleotide sequence or aligned gene segments), so you need to supply column names which has this purpose in your input data.

Usage

```
parse.cloneset(.filename, .nuc.seq, .aa.seq, .reads, .barcodes, .vgenes,
  .jgenes, .dgenes, .vend, .jstart, .dalignments, .vd.insertions,
  .dj.insertions, .total.insertions, .skip = 0, .sep = "\t")
```

Arguments

.filename	Path to the input file with cloneset data.
.nuc.seq	Name of the column with CDR3 nucleotide sequences.
.aa.seq	Name of the column with CDR3 amino acid sequences.
.reads	Name of the column with counts of reads for each clonotype.
.barcodes	Name of the column with counts of barcodes (UMI, events) for each clonotype.
.vgenes	Name of the column with names of aligned Variable gene segments.
.jgenes	Name of the column with names of aligned Joining gene segments.
.dgenes	Name of the column with names of aligned Diversity gene segments.
.vend	Name of the column with last positions of aligned V gene segments.
.jstart	Name of the column with first positions of aligned J gene segments.
.dalignments	Character vector of length two that names columns with D5' and D3' end positions.
.vd.insertions	Name of the column with VD insertions for each clonotype.
.dj.insertions	Name of the column with DJ insertions for each clonotype.
.total.insertions	Name of the column with total number of insertions for each clonotype.
.skip	How many lines from beginning to skip.
.sep	Separator character.

Value

Data frame with immune receptor repertoire data. See parse.file for more details.

See Also

parse.file

Examples

```
## Not run:
# Parse file in "~/mitcr/immdata1.txt" as a MiTCR file.
immdata1 <- parse.file("~/mitcr/immdata1.txt", 'mitcr')

## End(Not run)
```

parse.folder *Parse input table files with immune receptor repertoire data.*

Description

Load the TCR data from the file with the given filename to a data frame or load all files from the given folder to a list of data frames. The folder must contain onky files with the specified format. Input files could be either text files or archived with gzip ("filename.txt.gz") or bzip2 ("filename.txt.bz2"). For a general parser see parse.cloneset.

Parsers are available for: MiTCR ("mitcr"), MiTCR w/ UMIs ("mitcrbc"), MiGEC ("migec"), VDJtools ("vdjtools"), ImmunoSEQ ("immunoseq" or 'immunoseq2' for old and new formats respectively), MiXCR ("mixcr"), IMSEQ ("imseq") and tcR ("tcr", data frames saved with the 'repSave()' function).

Output of MiXCR should contain either all hits or best hits for each gene segment.

Output of IMSEQ should be generated with parameter "-on". In this case there will be no positions of aligned gene segments in the output data frame due to restrictions of IMSEQ output.

tcR's data frames should be saved with the 'repSave()' function.

Usage

```
parse.file(.filename,
.format = c('mitcr', 'mitcrbc', 'migec', 'vdjtools', 'immunoseq',
'mixcr', 'imseq', 'tcr'), ...)

parse.file.list(.filenames,
.format = c('mitcr', 'mitcrbc', 'migec', 'vdjtools', 'immunoseq',
'mixcr', 'imseq', 'tcr'), .namelist = NA)

parse.folder(.folderpath,
.format = c('mitcr', 'mitcrbc', 'migec', 'vdjtools', 'immunoseq',
'mixcr', 'imseq', 'tcr'), ...)

parse.mitcr(.filename)

parse.mitcrbc(.filename)

parse.migec(.filename)

parse.vdjtools(.filename)

parse.immunoseq(.filename)

parse.immunoseq2(.filename)

parse.immunoseq3(.filename)
```

```
parse.mixcr(.filename)

parse.imseq(.filename)

parse.tcr(.filename)

parse.migmap(.filename)
```

Arguments

`.folderpath`	Path to the folder with text cloneset files.
`.format`	String that specifies the input format.
`...`	Parameters passed to `parse.cloneset`.
`.filename`	Path to the input file with cloneset data.
`.filenames`	Vector or list with paths to files with cloneset data.
`.namelist`	Either NA or character vector of length `.filenames` with names for output data frames.

Value

Data frame with immune receptor repertoire data. Each row in this data frame corresponds to a clonotype. The data frame has following columns:

- "Umi.count" - number of barcodes (events, UMIs);

- "Umi.proportion" - proportion of barcodes (events, UMIs);

- "Read.count" - number of reads;

- "Read.proportion" - proportion of reads;

- "CDR3.nucleotide.sequence" - CDR3 nucleotide sequence;

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

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

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

- "D.gene" - names of aligned Diversity gene segments;

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

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

- "D5.end" - positions of D'5 end of aligned D gene segments (1-based);

- "D3.end" - positions of D'3 end of aligned D gene segments (1-based);

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

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

- "Total.insertions" - total number of inserted nucleotides (number of N-nucleotides at V-J junction for receptors with VJ recombination).

See Also

parse.cloneset, repSave, repLoad

Examples

```
## Not run:
# Parse file in "~/mitcr/immdata1.txt" as a MiTCR file.
immdata1 <- parse.file("~/mitcr/immdata1.txt", 'mitcr')
# Parse VDJtools file archive as .gz file.
immdata1 <- parse.file("~/mitcr/immdata3.txt.gz", 'vdjtools')
# Parse files "~/data/immdata1.txt" and "~/data/immdat2.txt" as MiGEC files.
immdata12 <- parse.file.list(c("~/data/immdata1.txt",
                               "~/data/immdata2.txt"), 'migec')
# Parse all files in "~/data/" as MiGEC files.
immdata <- parse.folder("~/data/", 'migec')

## End(Not run)
```

---

pca.segments *Perform PCA on segments frequency data.*

Description

Perform PCA on gene segments frequency data for V- and J-segments and either return pca object or plot the results.

Usage

```
pca.segments(.data, .cast.freq.seg = T, ..., .text = T, .do.plot = T)

pca.segments.2D(.data, .cast.freq.seg = T, ..., .text = T, .do.plot = T)
```

Arguments

.data	Either data.frame or a list of data.frame or a result obtained from the geneUsage function.
.cast.freq.seg	if T then apply codegeneUsage to the supplied data.
...	Further arguments passed to prcomp or geneUsage.
.text	If T then plot sample names in the resulting plot.
.do.plot	if T then plot a graphic, else return a pca object.

Value

If .do.plot is T than ggplot object; else pca object.

Examples

```
## Not run:
# Load the twins data.
data(twb)
# Plot a plot of results of PCA on V-segments usage.
pca.segments(twb, T, scale. = T)

## End(Not run)
```

`pca2euclid` *Compute the Euclidean distance among principal components.*

Description

Compute the Euclidean distance among principal components.

Usage

```
pca2euclid(.pcaobj, .num.comps = 2)
```

Arguments

`.pcaobj`	An object returned by `prcomp`.
`.num.comps`	On how many principal components compute the distance.

Value

Matrix of distances.

See Also

prcomp, pca.segments, repOverlap, permutDistTest

Examples

```
## Not run:
mat.ov <- repOverlap(AS_DATA, .norm = T)
mat.gen.pca <- pca.segments(AS_DATA, T, .genes = HUMAN_TRBV)
mat.ov.pca <- prcomp(mat.ov, scale. = T)
mat.gen.pca.dist <- pca2euclid(mat.gen.pca)
mat.ov.pca.dist <- pca2euclid(mat.ov.pca)
permutDistTest(mat.gen.pca.dist, list(<list of groups here>))
permutDistTest(mat.ov.pca.dist, list(<list of groups here>))

## End(Not run)
```

`permutDistTest` *Monte Carlo permutation test for pairwise and one-vs-all-wise within- and inter-group differences in a set of repertoires.*

Description

WARNING: this is an experimental procedure, work is still in progress.

Perform permutation tests of distances among groups for the given groups of samples and matrix of distances among all samples.

Usage

```
permutDistTest(.mat, .groups, .n = 1000, .fun = mean, .signif = 0.05,
  .plot = T, .xlab = "Values",
  .title = "Monte Carlo permutation testing of overlaps", .hjust = -0.1,
  .vjust = -4)
```

Arguments

`.mat`	Symmetric matrix of repertoire distances.
`.groups`	Named list with names of repertoires in groups.
`.n`	Number of permutations for each pair of group.
`.fun`	A function to apply to distances.
`.signif`	Significance level. Below this value hypotheses counts as significant.
`.plot`	If T than plot the output results. Else return them as a data frame.
`.xlab`	X lab label.
`.title`	Main title of the plot.
`.hjust`	Value for adjusting the x coordinate of p-value labels on plots.
`.vjust`	Value for adjusting the y coordinate of p-value labels on plots.

See Also

repOverlap, intersectClonesets, ozScore, pca2euclid

Examples

```
## Not run:
data(twb)
mat <- repOverlap(twb)
permutDistTest(mat, list(tw1 = c('Subj.A', 'Subj.B'), tw2 = c('Subj.C', 'Subj.D')))
permutDistTest(mat, list(tw1 = c('Subj.A', 'Subj.B'), tw2 = c('Subj.C', 'Subj.D')), .fun = median)

## End(Not run)
```

permutedf *Shuffling data frames.*

Description

Shuffle the given data.frame and order it by the Read.count column or un-shuffle a data frame and return it to the initial order.

Usage

```
permutedf(.data)

unpermutedf(.data)
```

Arguments

.data	MiTCR data.frame or list of such data frames.

Value

Shuffled data.frame or un-shuffled data frame if .data is a data frame, else list of such data frames.

rarefaction *Diversity evaluation using rarefaction.*

Description

Sequentially resample the given data with growing sample size the given data and compute mean number of unique clones. For more details on the procedure see "Details".

Usage

```
rarefaction(.data, .step = 30000, .quantile = c(0.025, 0.975),
  .extrapolation = 2e+05, .col = "Umi.count", .verbose = T)
```

Arguments

.data	Data frame or a list with data frames.
.step	Step's size.
.quantile	Numeric vector of length 2 with quantiles for confidence intervals.
.extrapolation	If N > 0 than perform extrapolation of all samples to the size of the max one +N reads or UMIs.
.col	Column's name from which choose frequency of each clone.
.verbose	if T then print progress bar.

Details

This subroutine is designed for diversity evaluation of repertoires. On each step it computes a mean unique clones from sample of fixed size using bootstrapping. Unique clones for each sample from bootstrap computed as a number of non-zero elements in a vector from multinomial distribution with input vector of probabilities from the `.col` column using function `rmultinom` with parameters n = .n, size = i * .step, prob = .data[, .col] (i is an index of current iteration) and choosing for lower and upper bound `quantile` bounds of the computed distribution of unique clones.

Value

Data frame with first column for sizes, second columns for the first quantile, third column for the mean, fourth columns for the second quantile, fifth columns for the name of subject.

See Also

vis.rarefaction rmultinom

Examples

```
## Not run:
rarefaction(immdata, .col = "Read.count")

## End(Not run)
```

`repDiversity` *General function for the repertoire diversity estimation.*

Description

General interface to all cloneset diversity functions.

Usage

```
repDiversity(.data, .method = c("chao1", "gini.simp", "inv.simp", "gini",
  "div", "entropy"), .quant = c("read.count", "umi.count", "read.prop",
  "umi.prop"), .q = 5, .norm = F, .do.norm = NA, .laplace = 0)
```

Arguments

`.data`	Cloneset or a list of clonesets.
`.method`	Which method to use for the diversity estimation. See "Details" for methods.
`.quant`	Which column to use for the quantity of clonotypes: "read.count" for the "Read.count" column, "umi.count" for the "Umi.count" column, "read.prop" for the "Read.proportion" column, "umi.prop" for the "Umi.proportion" column.
`.q`	q-parameter for the Diversity index.
`.norm`	If T than compute the normsalised entropy.

.do.norm	One of the three values - NA, T or F. If NA than check for distrubution (sum(.data) == 1) and normalise it with the given laplace correction value if needed. if T then do normalisation and laplace correction. If F than don't do normalisaton and laplace correction.
.laplace	Value for Laplace correction.

Details

You can see a more detailed description for each diversity method at diversity.

Parameter .method can have one of the following value each corresponding to the specific method:

- "div" for the true diversity, or the effective number of types (basic function diversity).

- "inv.simp" for the inverse Simpson index (basic function inverse.simpson).

- "gini" for the Gini coefficient (basic function gini).

- "gini.simp" for the Gini-Simpson index (basic function gini.simpson).

- "chao1" for the Chao1 estimator (basic function chao1).

- "entropy" for the Shannon entropy measure (basic function entropy).

See Also

diversity, entropy

Examples

```
## Not run:
data(twb)
twb.div <- repDiversity(twb, "chao1", "read.count")

## End(Not run)
```

---

repLoad *Parse input files or folders with immune receptor repertoire data.*

Description

Load the immune receptor repertoire data from the given input: either a file name, a list of file names, a name of the folder with repertoire files, or a list of folders with repertoire files. The folder / folders must contain only files with the specified format. Input files could be either text files or archived with gzip ("filename.txt.gz") or bzip2 ("filename.txt.bz2"). For a general parser of table files with cloneset data see parse.cloneset.

Parsers are available for: MiTCR ("mitcr"), MiTCR w/ UMIs ("mitcrbc"), MiGEC ("migec"), VDJtools ("vdjtools"), ImmunoSEQ ("immunoseq" or 'immunoseq2' for old and new formats respectively), MiXCR ("mixcr"), IMSEQ ("imseq") and tcR ("tcr", data frames saved with the 'repSave()' function).

Output of MiXCR should contain either all hits or best hits for each gene segment.

Output of IMSEQ should be generated with parameter "-on". In this case there will be no positions of aligned gene segments in the output data frame due to restrictions of IMSEQ output.

tcR's data frames should be saved with the 'repSave()' function.

For details on the tcR data frame format see parse.file.

Usage

```
repLoad(.path, .format = c("mitcr", "migec"))
```

Arguments

.path	Character vector with path to files and / or folders.
.format	String that specifies the input format.

See Also

parse.file

Examples

```
## Not run:
datalist <- repLoad(c("file1.txt", "folder_with_files1", "another_folder"), "mixcr")

## End(Not run)
```

---

repOverlap *General function for the repertoire overlap evaluation.*

Description

General interface to all cloneset overlap functions.

Usage

```
repOverlap(.data, .method = c("exact", "hamm", "lev", "jaccard", "morisita",
  "tversky", "overlap", "horn"), .seq = c("nuc", "aa"),
  .quant = c("read.count", "umi.count", "read.prop", "umi.prop"),
  .vgene = F, .norm = T, .a = 0.5, .b = 0.5, .do.unique = T,
  .verbose = T)
```

Arguments

.data	List of clonesets.
.method	Which method to use for the overlap evaluation. See "Details" for methods.
.seq	Which clonotype sequences to use for the overlap: "nuc" for "CDR3.nucleotide.sequence", "aa" for "CDR3.amino.acid.sequence".

.quant	Which column to use for the quantity of clonotypes: "read.count" for the "Read.count" column, "umi.count" for the "Umi.count" column, "read.prop" for the "Read.proportion" column, "umi.prop" for the "Umi.proportion" column. Used in "morisita" and "horn".
.vgene	If T than use V genes in computing shared or similar clonotypes. Used in all methods.
.norm	If T than compute the normalised number of shared clonotypes. Used in "exact".
.a, .b	Alpha and beta parameters for "tversky". Default values gives the Jaccard index measure.
.do.unique	If T than remove duplicates from the input data, but add their quantities to their clones.
.verbose	If T than output the data processing progress bar.

Details

You can see a more detailed description for each overlap method at intersectClonesets and similarity.

Parameter .method can have one of the following value each corresponding to the specific method:

- "exact" for the shared number of clonotypes (basic function intersectClonesets(..., .type = "..e")).

- "hamm" for the number of similar clonotypes by the Hamming distance (basic function intersectClonesets(..., .type

- "lev" for the number of similar clonotypes by the Levenshtein distance (basic function intersectClonesets(..., .type =

- "jaccard" for the Jaccard index (basic function jaccard.index).

- "morisita" for the Morisita's overlap index (basic function morisita.index).

- "tversky" for the Tversky index (basic function tversky.index).

- "overlap" for the overlap coefficient (basic function overlap.coef).

- "horn" for the Horn's index (basic function horn.index).

See Also

intersectClonesets, similarity, repDiversity

Examples

```
## Not run:
data(twb)
repOverlap(twb, "exact", .seq = "nuc", .vgene = F)
repOverlap(twb, "morisita", .seq = "aa", .vgene = T, .quant = "umi.count")
ov <- repOverlap(twb)
vis.pca(prcomp(ov, scale. = T), list(A = c(1, 2), B = c(3, 4)))

## End(Not run)
```

repSave *Save tcR data frames to disk as text files or gzipped text files.*

Description

Save repertoire files to either text files or gzipped text files. You can read them later by `repLoad` function with `.format = "tcr"`.

Usage

```
repSave(.data, .format = c("txt", "gz"), .names = "", .folder = "./")
```

Arguments

`.data`	Either tcR data frame or a list of tcR data frames.
`.format`	"txt" for simple tab-delimited text tables, "gz" for compressed (gzipped) tables.
`.names`	Names of output files. By default it's an empty string so names will be taken from names of the input list.
`.folder`	Path to the folder with output files.

See Also

repLoad

resample *Resample data frame using values from the column with number of clonesets.*

Description

Resample data frame using values from the column with number of clonesets. Number of clonestes (i.e., rows of a MiTCR data frame) are reads (usually the "Read.count" column) or UMIs (i.e., barcodes, usually the "Umi.count" column).

Usage

```
resample(.data, .n = -1, .col = "read.count")
```

Arguments

`.data`	Data frame with the column `.col` or list of such data frames.
`.n`	Number of values / reads / UMIs to choose.
`.col`	Which column choose to represent quanitites of clonotypes. See "Details".

Details

`resample`. Using multinomial distribution, compute the number of occurences for each cloneset, than remove zero-number clonotypes and return resulting data frame. Probabilities for `rmultinom` for each cloneset is a percentage of this cloneset in the `.col` column. It's a some sort of simulation of how clonotypes are chosen from the organisms. For now it's not working very well, so use `downsample` instead.

`downsample`. Choose `.n` clones (not clonotypes!) from the input repertoires without any probabilistic simulation, but exactly computing each choosed clones. Its output is same as for `resample` (repertoires), but is more consistent and biologically pleasant.

Value

Data frame with `sum(.data[, .col]) == .n`.

See Also

rmultinom

Examples

```
## Not run:
# Get 100K reads (not clones!).
immdata.1.100k <- resample(immdata[[1]], 100000, .col = "read.count")

## End(Not run)
```

---

revcomp *DNA reverse complementing and translation.*

Description

Functions for DNA reverse complementing and translation.

Usage

```
revcomp(.seq)

bunch.translate(.seq, .two.way = T)
```

Arguments

`.seq`	Vector of nucleotide sequences.
`.two.way`	if T then translate sequences from both ends (output differes for out-of-frame sequences).

Value

Vector of corresponding revese complemented or aminoacid sequences.

reverse.string *Reverse given character vector by the given n-plets.*

Description

Reverse given character vector by the given n-plets.

Usage

```
reverse.string(.seq, .n = 1)
```

Arguments

.seq	Sequences.
.n	By which n-plets we should reverse the given strings.

Value

Reversed strings.

Examples

```
reverse.string('abcde')  # => "edcba"
reverse.string('abcde', 2)  # => "debca"
```

sample.clones *Get a random subset from a data.frame.*

Description

Sample rows of the given data frame with replacement.

Usage

```
sample.clones(.data, .n, .replace = T)
```

Arguments

.data	Data.frame or a list with data.frames
.n	Sample size if integer. If in bounds [0;1] than percent of rows to extract. "1" is a percent, not one row!
.replace	if T then choose with replacement, else without.

Value

Data.frame of nrow .n or a list with such data.frames.

sample2D *Get a sample from matrix with probabilities.*

Description

Get a sample from matrix or data frame with pair-wise probabilities.

Usage

```
sample2D(.table, .count = 1)
```

Arguments

.table	Numeric matrix or data frame with probabilities and columns and rows names.
.count	Number of sample to fetch.

Value

Character matrix with nrow == .count and 2 columns. row[1] in row.names(.table), row[2] in colnames(.table).

segments.alphabets *Alphabets of TCR and Ig gene segments.*

Description

Character vector with names for segments. With tcR we provided alphabets for all alpha, beta, gamma and delta chains gene segments.

Usage

```
HUMAN_TRAV

HUMAN_TRAJ

HUMAN_TRBV

HUMAN_TRBD

HUMAN_TRBJ

HUMAN_TRBV_MITCR

HUMAN_TRBV_ALS
```

HUMAN_TRGV

HUMAN_TRGJ

HUMAN_TRDV

HUMAN_TRDD

HUMAN_TRDJ

MOUSE_TRBV

MOUSE_TRBJ

MOUSE_TRAV

MOUSE_TRAJ

MOUSE_IGKV

MOUSE_IGKJ

MOUSE_IGHV

MOUSE_IGHD

MOUSE_IGHJ

MACMUL_TRBV

MACMUL_TRBJ

HUMAN_IGHV

HUMAN_IGHD

HUMAN_IGHJ

HUMAN_IGLV

HUMAN_IGLJ

MOUSE_IGLJ

MOUSE_IGLV

Format

Each <SPECIES>_<GENES> is a character vector. <SPECIES> is an identifier of species, <GENES> is concatenated three identifiers of cell type ("TR**" for TCR, "IG**" for Ig), chain (e.g., "**A*" for alpha chains) and gene segment ("***V" for V(ariable) gene segment, "***J" for J(oining) gene segment, "***D" for D(iversity) gene segment).

Examples

```
## Not run:
HUMAN_TRBV[1]  #  => "TRBV10-1"

## End(Not run)
```

`segments.list` *Segment data.*

Description

`segments` is a list with 5 data frames with data of human alpha-beta chain segments. Elements names as "TRAV", "TRAJ", "TRBV", "TRVJ", "TRVD". Each data frame consists of 5 columns:

- V.allelles / J.allelles / D.allelles - character column with names of V/D/J-segments.

- CDR3.position - position in the full nucleotide segment sequence where CDR3 starts.

- Full.nucleotide.sequence - character column with segment CDR1-2-3 sequence.

- Nucleotide.sequence - character column with segment CDR3 sequences.

- Nucleotide.sequence.P - character column with segment CDR3 sequences with P-insertions.

Format

`genesegments` is a list with data frames.

Examples

```
## Not run:
data(genesegments)
genesegments$Nucleotide.sequence[segments$TRBV[,1] == "TRBV10-1"]

## End(Not run)
```

`set.group.vector` *Set group attribute for vertices of a mutation network*

Description

asdasd

Usage

```
set.group.vector(.G, .attr.name, .groups)

get.group.names(.G, .attr.name, .V = V(.G), .paste = T)
```

Arguments

`.G`	Mutation network.
`.attr.name`	Name of the new vertex attribute.
`.groups`	List with integer vector with indices of subjects for each group.
`.V`	Indices of vertices.
`.paste`	if T then return character string with concatenated group names, else return list with character vectors with group names.

Value

igraph object with new vertex attribute `.attr.name` with binary strings for `set.group.vector`. Return character vector for `get.group.names`.

Examples

```
## Not run:
data(twb)
twb.shared <- shared.repertoire(twb)
G <- mutation.network(twb.shared)
G <- set.group.vector(G, "twins", list(A = c(1,2), B = c(3,4)))  # <= refactor this
get.group.names(G, "twins", 1)       # "A|B"
get.group.names(G, "twins", 300)     # "A"
get.group.names(G, "twins", 1, F)    # list(c("A", "B"))
get.group.names(G, "twins", 300, F)  # list(c("A"))
# Because we have only two groups, we can assign more readable attribute.
V(G)$twin.names <- get.group.names(G, "twins")
V(G)$twin.names[1]  # "A|B"
V(G)$twin.names[300]  # "A"

## End(Not run)
```

set.pb — *Simple functions for manipulating progress bars.*

Description

Set the progress bar with the given length (from zero) or add value to the created progress bar.

Usage

```
set.pb(.max)

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

Arguments

.max	Length of the progress bar.
.pb	Progress bar object.
.value	Value to add to the progress bar.

Value

Progress bar (for set.pb) or length-one numeric vector giving the previous value (for add.pb).

set.people.vector — *Set and get attributes of a mutation network related to source people.*

Description

Set vertice attributes 'people' and 'npeople' for every vertex in the given graph. Attribute 'people' is a binary string indicating in which repertoire sequence are found. Attribute 'npeople' is a integer indicating number of repertoires, in which this sequence has been found.

Usage

```
set.people.vector(.G, .shared.rep)

get.people.names(.G, .V = V(.G), .paste = T)
```

Arguments

.G	Mutation network.
.shared.rep	Shared repertoire.
.V	Indices of vertices.
.paste	If TRUE than concatenate people names to one string, else get a character vector of names.

Value

New graph with 'people' and 'npeople' vertex attributes or character vector of length .V or list of length .V.

Examples

```
## Not run:
data(twb)
twb.shared <- shared.repertoire(twb)
G <- mutation.network(twb.shared)
get.people.names(G, 300, T)  # "Subj.A|Subj.B"
get.people.names(G, 300, F)  # list(c("Subj.A", "Subj.B"))

## End(Not run)
```

`set.rank` *Set new columns "Rank" and "Index".*

Description

Set new columns "Rank" and "Index":

set.rank <==> .data$Rank = rank(.data[, .col], ties.method = 'average')

set.index <==> .data$Index = 1:nrow(.data) in a sorted data frame by `.col`

Usage

```
set.rank(.data, .col = "Read.count")
```

Arguments

`.data`	Data frame or list with data frames.
`.col`	Character vector with name of the column to use for ranking or indexing.

Value

Data frame with new column "Rank" or "Index" or list with such data frames.

shared.repertoire *Shared TCR repertoire managing and analysis*

Description

Generate a repertoire of shared sequences - sequences presented in more than one subject. If sequence is appeared more than once in the one repertoire, than only the first appeared one will be choosed for a shared repertoire.

`shared.repertoire` - make a shared repertoire of sequences from the given list of data frames.

`shared.matrix` - leave columns, which related to the count of sequences in people, and return them as a matrix. I.e., this functions will remove such columns as 'CDR3.amino.acid.sequence', 'V.gene', 'People'.

Usage

```
shared.repertoire(.datalist, .type = 'avrc', .min.ppl = 1, .head = -1,
                  .clear = T, .verbose = T, .by.col = '', .sum.col = '',
                  .max.ppl = length(.datalist))

shared.matrix(.shared.rep)
```

Arguments

`.datalist`	List with data frames.
`.type`	String of length 4 denotes how to create a shared repertoire. See "Details" for more information. If supplied, than parameters `.by.col` and `.sum.col` will be ignored. If not supplied, than columns in `.by.col` and `.sum.col` will be used.
`.min.ppl`	At least how many people must have a sequence to leave this sequence in the shared repertoire.
`.head`	Parameter for the `head` function, applied to all data frames before clearing.
`.clear`	if T then remove all sequences which have symbols "~" or "*" (i.e., out-of-frame sequences for amino acid sequences).
`.verbose`	if T then output progress.
`.by.col`	Character vector with names of columns with sequences and their parameters (like segment) for using for creating a shared repertoire.
`.sum.col`	Character vector of length 1 with names of the column with count, percentage or any other numeric chaaracteristic of sequences for using for creating a shared repertoire.
`.max.ppl`	At most how many people must have a sequence to leave this sequence in the shared repertoire.
`.shared.rep`	Shared repertoire.

Details

Parameter `.type` is a string of length 4, where:

1. First character stands either for the letter 'a' for taking the "CDR3.amino.acid.sequence" column or for the letter 'n' for taking the "CDR3.nucleotide.sequence" column.
2. Second character stands whether or not take the V.gene column. Possible values are '0' (zero) stands for taking no additional columns, 'v' stands for taking the "V.gene" column.
3. Third character stands for using either UMIs or reads in choosing the column with numeric characterisitc (see the next letter).
4. Fourth character stands for name of the column to choose as numeric characteristic of sequences. It depends on the third letter. Possible values are "c" for the "Umi.count" (if 3rd character is "u") / "Read.count" column (if 3rd character is "r"), "p" for the "Umi.proportion" / "Read.proportion" column, "r" for the "Rank" column or "i" for the "Index" column. If "Rank" or "Index" isn't in the given repertoire, than it will be created using `set.rank` function using "Umi.count" / "Read.count" column.

Value

Data frame for `shared.repertoire`, matrix for `shared.matrix`.

See Also

shared.representation, set.rank

Examples

```
## Not run:
# Set "Rank" column in data by "Read.count" column.
# This is doing automatically in shared.repertoire() function
# if the "Rank" column hasn't been found.
immdata <- set.rank(immdata)
# Generate shared repertoire using "CDR3.amino.acid.sequence" and
# "V.gene" columns and with rank.
imm.shared.av <- shared.repertoire(immdata, 'avrc')

## End(Not run)
```

`spectratyping` *Spectratype plot.*

Description

General function for making a spectratyping plots.

Usage

```
spectratyping(.data, .column = "VD.insertions", .by.alphabet = "Vb",
  .do.legend = T, .draw.only.legend = F, .legend.ncol = -1, .nrow = 2,
  .by.col = "", .sum.col = "Percentage", .other = F, .log = F,
  .verbose = T)
```

Arguments

.data	mitcr List with data frames.
.column	Character vector with name of the column with numeric characteristic.
.by.alphabet	Either 'Va/b', 'Ja/b' or an alphabet.
.do.legend	if T then plot a legend.
.draw.only.legend	if T then plot only a legend without plots.
.legend.ncol	Number of columns in the legend. If -1 than function will try to predict number of columns.
.nrow	Rows of grid of plots.
.by.col	Character vector with name of the column by which divide the given data frames.
.sum.col	Which column use for sum.
.other	if T then include in the result plot values which isn't in the given alphabet.
.log	if T then scale y-axis by log10.
.verbose	if T then print messages about state of the process.

Details

For each element in .data do: for each factor in .by.col which is in .by.alphabet, compute histogram of .column and then plot stacked histogram of distributions of .by.col for each factor.

Value

ggplot object.

Examples

```
## Not run:
# Spectratyping of distribution of length of CDR3 nucleotide sequences
# by V-beta-segments.
immdata <- lapply(immdata, function (x) {
             x$Length <- nchar(x$CDR3.nucleotide.sequence)
             x
           } )
spectratyping(immdata, 'Length', 'Vb')
# Spectratyping of distribution of Total insertions
# by J-beta-segments.
spectratyping(immdata, 'Total.insertions', 'Jb')

## End(Not run)
```

`startmitcr` *Start MiTCR directly from the package.*

Description

Start the MiTCR tools directly from the package with given settings.

Usage

```
startmitcr(.input = "", .output = "", ..., .file.path = "",
  .mitcr.path = "~/programs/", .mem = "4g")
```

Arguments

`.input, .output`	Input and output files.
`...`	Specify input and output files and arguments of the MITCR without first '-' to run it.
`.file.path`	Path prepending to `.input` and `.output`. If input and output is empty, but .file.path is specified, than process all files from the folder `.file.path`
`.mitcr.path`	Path to MiTCR .jar file.
`.mem`	Volume of memory available to MiTCR.

Details

Don't use spaces in paths! You should have insalled JDK 1.7 to make it works.

Examples

```
## Not run:
# Equal to
# java -Xmx8g -jar ~/programs/mitcr.jar -pset flex
#      -level 2 ~/data/raw/TwA1_B.fastq.gz ~/data/mitcr/TwA1_B.txt
startmitcr('raw/TwA1_B.fastq.gz', 'mitcr/TwA1_B.txt', .file.path = '~/data/',
  pset = 'flex', level = 1, 'debug', .mitcr.path = '~/programs/', .mem = '8g')

## End(Not run)
```

`tailbound.proportion` *Proportions of specifyed subsets of clones.*

Description

Get a specifyed subset of the given data and compute which proportion in counts it has comparing to the overall count.

`tailbound.proportion` - subset by the count;

`top.proportion` - subset by rank (top N clones);

`clonal.proportion` - subset by a summary percentage (top N clones which in sum has the given percentage).

Usage

```
tailbound.proportion(.data, .bound = 2, .col = 'Read.count')

top.proportion(.data, .head = 10, .col = 'Read.count')

clonal.proportion(.data, .perc = 10, .col = 'Read.count')
```

Arguments

`.data`	Data frame or a list with data frames.
`.bound`	Subset the `.data` by `.col` <= `.bound`.
`.col`	Column's name with counts of sequences.
`.head`	How many top values to choose - parameter to the `.head` function.
`.perc`	Percentage (0 - 100).

Value

For `tailbound.proportion` - numeric vector of percentage.

For `top.proportion` - numeric vector of percentage for top clones. For `clonal.proportion` - vector or matrix with values for number of clones, occupied percentage and proportion of the chosen clones to the overall count of clones.

See Also

vis.top.proportions

Examples

```
## Not run:
                                # How many clones fill up approximately
clonal.proportion(immdata, 25)  # the 25% of the sum of values in 'Read.count'?

                                # What proportion of the top-10 clones' reads
vis.top.proportions(immdata)  # Plot this proportions.

                                # What proportion of sequences which
                                # has 'Read.count' <= 100 to the
tailbound.proportion(immdata, 100)  # overall number of reads?

## End(Not run)
```

`top.cross` *Perform sequential cross starting from the top of a data frame.*

Description

`top.cross` - get top crosses of the given type between each pair of the given data.frames with `top.cross` function.

`top.cross.vec` - get vector of cross values for each top with the `top.cross.vec` function.

`top.cross.plot` - plot a plots with result with the `top.cross.plot` function.

Usage

```
top.cross(.data, .n = NA, .data2 = NULL, .type = 'ave', .norm = F, .verbose = T)

top.cross.vec(.top.cross.res, .i, .j)

top.cross.plot(.top.cross.res, .xlab = 'Top X clonotypes',
               .ylab = 'Normalised number of shared clonotypes', .nrow = 2,
               .legend.ncol = 1, .logx = T, .logy = T)
```

Arguments

`.data`	Either list of data.frames or a data.frame.
`.n`	Integer vector of parameter appled to the head function; same as .n in the top.fun function. See "Details" for more information.
`.data2`	Second data.frame or NULL if .data is a list.
`.type`	Parameter .type to the `tcR::intersect` function.
`.norm`	Parameter .norm to the `tcR::intersect` function.
`.verbose`	if T then plot a progress bar.
`.top.cross.res`	Result from the `top.cross` function.
`.i, .j`	Coordinate of a cell in each matrix.

.xlab	Name for a x-lab.
.ylab	Name for a y-lab.
.nrow	Number of rows of sub-plots in the output plot.
.legend.ncol	Number of columns in the output legend.
.logx	if T then transform x-axis to log-scale.
.logy	if T then transform y-axis to log-scale.

Details

Parameter .n can have two possible values. It could be either integer vector of numbers (same as in the top.fun function) or NA and then it will be replaced internally by the value .n <- seq(5000, min(sapply(.data, nrow

Value

top.cross - return list for each element in .n with intersection matrix (from tcR::intersectClonesets).

top.cross.vec - vector of length .n with .i:.j elements of each matrix.

top.cross.plot - grid / ggplot object.

See Also

intersect

Examples

```
## Not run:
immdata.top <- top.cross(immdata)
top.cross.plot(immdata.top)

## End(Not run)
```

---

top.fun *Get samples from a repertoire slice-by-slice or top-by-top and apply function to them.*

Description

Functions for getting samples from data frames either by consequently applying head functions (top.fun) or by getting equal number of rows in the moving window (slice.fun) and applying specified function to this samples.

Usage

```
top.fun(.data, .n, .fun, ..., .simplify = T)

slice.fun(.data, .size, .n, .fun, ..., .simplify = T)
```

Arguments

.data	Data.frame, matrix, vector or any enumerated type or a list of this types.
.n	Vector of values passed to head function for top.fun or the number of slices for slice.fun.
.fun	Funtions to apply to every sample subset. First input argument is a data.frame, others are passed as ....
...	Additional parameters passed to the .fun.
.simplify	if T then try to simplify result to a vector or to a matrix if .data is a list.
.size	Size of the slice for sampling for slice.fun.

Value

List of length length(.n) for top.fun or .n for slice.fun.

Examples

```
## Not run:
# Get entropy of V-usage for the first 1000, 2000, 3000, ... clones.
res <- top.fun(immdata[[1]], 1000, entropy.seg)
# Get entropy of V-usage for the interval of clones with indices [1,1000], [1001,2000], ...
res <- top.fun(immdata[[1]], 1000, entropy.seg)

## End(Not run)
```

---

twinsdata *Twins alpha-beta chain data*

Description

twa.rda, twb.rda - data frames with downsampled to the 10000 most abundant clonesets and 4 samples data of twins data (alpha and beta chains). Link: http://labcfg.ibch.ru/tcr.html

Format

twa and twb are lists of 4 data frames with 10000 row in each.

Examples

```
## Not run:
data(twa)
data(twb)

## End(Not run)
```

vis.clonal.dynamics *Visualise clonal dynamics among time points.*

Description

Visualise clonal dynamics (i.e., changes in frequency or count) with error bars of given clones among time points.

Usage

```
vis.clonal.dynamics(.changed, .lower, .upper, .log = T)
```

Arguments

.changed	Result from the find.clonotypes function, i.e. data frame with first columns with sequences (nucleotide or amino acid) and other columns are columns with frequency / count for each time point for each clone.
.lower	Similar to .changed but values are lower bound for clonal count / frequency.
.upper	Similar to .changed but values are upper bound for clonal count / frequency.
.log	if T then log-scale y-axis.

Value

ggplot object.

vis.clonal.space *Visualise occupied by clones homeostatic space among Samples or groups.*

Description

Visualise which clones how much space occupy.

Usage

```
vis.clonal.space(.clonal.space.data, .groups = NULL)
```

Arguments

.clonal.space.data	Data from the fclonal.space.homeostasis function.
.groups	List of named character vector with names of Samples in .clonal.space.data for grouping them together.

Value

ggplot object.

See Also

clonal.space.homeostasis

`vis.count.len` *Plot a histogram of lengths.*

Description

Plot a histogram of distribution of lengths of CDR3 nucleotide sequences. On y-axis are sum of read counts for each length.

Usage

```
vis.count.len(.data, .ncol = 3, .name = "", .col = "Read.count")
```

Arguments

`.data`	Data frame with columns 'CDR3.nucleotide.sequence' and 'Read.count' or list with such data frames.
`.ncol`	If .data is a list, than number of columns in a grid of histograms for each data frame in `.data`. Else not used.
`.name`	Title for this plot.
`.col`	Name of the column to use in computing the lengths distribution.

Details

If `.data` is a data frame, than one histogram will be plotted. Is `.data` is a list, than grid of histograms will be plotted.

Value

ggplot object.

Examples

```
## Not run:
load('immdata.rda')
# Plot one histogram with main title.
vis.count.len(immdata[[1]], 'Main title here')
# Plot a grid of histograms with 2 columns.
vis.count.len(immdata, 2)

## End(Not run)
```

vis.gene.usage *Histogram of segments usage.*

Description

Plot a histogram or a grid of histograms of V- / J-usage.

Usage

```
vis.gene.usage(.data, .genes = NA, .main = "Gene usage", .ncol = 3,
  .coord.flip = F, .dodge = F, .labs = c("Gene", "Frequency"), ...)
```

Arguments

.data	Mitcr data frame or a list with mitcr data frames.
.genes	Gene alphabet passed to geneUsage.
.main	Main title of the plot.
.ncol	Number of columns in a grid of histograms if .data is a list and .dodge is F.
.coord.flip	if T then flip coordinates.
.dodge	If .data is a list, than if this is T plot V-usage for all data frames to the one histogram.
.labs	Character vector of length 2 with names for x-axis and y-axis.
...	Parameter passed to geneUsage. By default the function compute V-usage or J-usage for beta chains w/o using read counts and w/ "Other" segments.

Value

ggplot object.

Examples

```
## Not run:
# Load your data.
load('immdata.rda')
# Compute V-usage statistics.
imm1.vs <- geneUsage(immdata[[1]], HUMAN_TRBV)
vis.V.usage(immdata, HUMAN_TRBV, .main = 'Immdata V-usage [1]', .dodge = T)
# Plot a histogram for one data frame using all gene segment data from V.gene column.
vis.V.usage(imm1.vs, NA, .main = 'Immdata V-usage [1]')
# Plot a grid of histograms - one histogram for V-usage for each data frame in .data.
vis.V.usage(immdata, HUMAN_TRBV, .main = 'Immdata V-usage', .dodge = F, .other = F)

## End(Not run)
```

`vis.group.boxplot` *Boxplot for groups of observations.*

Description

Plot boxplots for each group.

Usage

```
vis.group.boxplot(.data, .groups = NA, .labs = c("V genes", "Frequency"),
  .title = "", .rotate.x = T, .violin = T, .notch = F, ...)
```

Arguments

`.data`	Either a matrix with colnames and rownames specifyed or a data frame with the first column of strings for row names and other columns stands for values.
`.groups`	Named list with character vectors for names of elements for each group. If NA than each member is in the individual group.
`.labs`	Labs names. Character vector of length 1 (for naming both axis with same name) or 2 (first elements stands for x-axis).
`.title`	Main title of the plot.
`.rotate.x`	if T then rotate x-axis.
`.violin`	If T then plot a violin plot.
`.notch`	"notch" parameter to the `geom_boxplot` ggplo2 function.
`...`	Parameters passed to `melt`, applied to `.data` before plotting in `vis.group.boxplot`.

Value

ggplot object.

Examples

```
## Not run:
names(immdata)  # "A1" "A2" "B1" "B2" "C1" "C2"
# Plot a boxplot for V-usage for each plot
# three boxplots for each group.
vis.group.boxplot(freq.Vb(immdata),
   list(A = c('A1', 'A2'), B = c('B1', 'B2'), C = c('C1', 'C2')),
   c('V segments', 'Frequency'))

data(twb)
ov <- repOverlap(twb)
sb <- matrixSubgroups(ov, list(tw1 = c('Subj.A', 'Subj.B'), tw2 = c('Subj.C', 'Subj.D')));
vis.group.boxplot(sb)

## End(Not run)
```

`vis.heatmap` *Heatmap.*

Description

Plot a heatmap from a matrix or a data.frame

Usage

```
vis.heatmap(.data, .title = "Number of shared clonotypes",
  .labs = c("Sample", "Sample"), .legend = "Shared clonotypes",
  .na.value = NA, .text = T, .scientific = FALSE, .signif.digits = 4,
  .size.text = 4, .no.legend = F, .no.labs = F)
```

Arguments

`.data`	Either a matrix with colnames and rownames specifyed or a data.frame with the first column of strings for row names and other columns stands for values.
`.title`	Main title of the plot.
`.labs`	Labs names. Character vector of length 2 (for naming x-axis and y-axis).
`.legend`	Title for the legend.
`.na.value`	Replace NAs with this values.
`.text`	if T then print `.data` values at tiles.
`.scientific`	If T then force show scientific values in the heatmap plot.
`.signif.digits`	Number of significant digits to show. Default - 4.
`.size.text`	Size for the text in the cells of the heatmap, 4 by default.
`.no.legend`	If T than remove the legend from the plot.
`.no.labs`	If T than remove x / y labels names from the plot.

Value

ggplot object.

Examples

```
## Not run:
# Load your data.
load('immdata.rda')
# Perform cloneset overlap by amino acid sequences with V-segments.
imm.av <- repOverlap(immdata, .seq = 'aa', .vgene = T)
# Plot a heatmap.
vis.heatmap(imm.av, .title = 'Immdata - (ave)-intersection')

## End(Not run)
```

`vis.kmer.histogram` *Plot of the most frequent kmers.*

Description

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

Usage

```
vis.kmer.histogram(.kmers, .head = 100, .position = c("stack", "dodge",
  "fill"))
```

Arguments

`.kmers`	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`.

See Also

`get.kmers`

Examples

```
## Not run:
# Load necessary data and package.
library(gridExtra)
load('immdata.rda')
# Get 5-mers.
imm.km <- get.kmers(immdata)
# Plots for kmer proportions in each data frame in immdata.
p1 <- vis.kmer.histogran(imm.km, .position = 'stack')
p2 <- vis.kmer.histogran(imm.km, .position = 'fill')
grid.arrange(p1, p2)

## End(Not run)
```

vis.logo *Logo - plots for amino acid and nucletide profiles.*

Description

Plot logo-like graphs for visualising of nucleotide or amino acid motif sequences / profiles.

Usage

```
vis.logo(.data, .replace.zero.with.na = T, .jitter.width = 0.01,
  .jitter.height = 0.01, .dodge.width = 0.15)
```

Arguments

`.data`	Output from the `kmer.profile` function.
`.replace.zero.with.na`	if T then replace all zeros with NAs, therefore letters with zero frequency wont appear at the plot.
`.jitter.width, .jitter.height, .dodge.width`	Parameters to `position_jitterdodge` for aligning text labels of letters.

Value

ggplot2 object

Examples

```
## Not run:
d <- kmer.profile(c('CASLL', 'CASSQ', 'CASGL'))
vis.logo(d)

## End(Not run)
```

vis.number.count *Plot a histogram of counts.*

Description

Plot a histogram of distribution of counts of CDR3 nucleotide sequences. On y-axis are number of counts.

Usage

```
vis.number.count(.data, .ncol = 3,
  .name = "Histogram of clonotypes read counts", .col = "Read.count")
```

Arguments

.data	Cloneset data frame or a list of clonesets.
.ncol	If .data is a list, than number of columns in a grid of histograms for each data frame in .data. Else not used.
.name	Title for this plot.
.col	Name of the column with counts.

Details

If .data is a data frame, than one histogram will be plotted. Is .data is a list, than grid of histograms will be plotted.

Value

ggplot object.

Examples

```
## Not run:
load('immdata.rda')
# Plot one histogram with main title.
vis.number.count(immdata[[1]], 'Main title here')
# Plot a grid of histograms with 2 columns.
vis.number.count(immdata, 2)

## End(Not run)
```

---

vis.pca *PCA result visualisation*

Description

Plot the given pca results with colour divided by the given groups.

Usage

```
vis.pca(.data, .groups = NA)
```

Arguments

.data	Result from prcomp() function or a data frame with two columns 'First' and 'Second' stands for the first PC and the second PC.
.groups	List with names for groups and indices of the group members. If NA than each member is in the individual group.

Value

ggplot object.

vis.radarlike *Radar-like / spider-like plots.*

Description

Plot a grid of radar(-like) plots for visualising a distance among objects.

Usage

```
vis.radarlike(.data, .ncol = 3, .expand = c(0.25, 0))
```

Arguments

.data	Square data frame or matrix with row names and col names stands for objects and values for distances.
.ncol	Number of columns in the grid.
.expand	Interger vector of length 2, for `scale_y_continous(expand = .expand)` function.

See Also

repOverlap, js.div

Examples

```
## Not run:
load('immdata.rda')
# Compute Jensen-Shannon divergence among V-usage of repertoires.
imm.js <- js.div.seg(immdata, .verbose = F)
# Plot it.
vis.radarlike(imm.js)

## End(Not run)
```

vis.rarefaction *Rarefaction statistics visualisation.*

Description

Plot a line with mean unique clones.

Usage

```
vis.rarefaction(.muc.res, .groups = NULL, .log = F)
```

Arguments

.muc.res	Output from the muc function.
.groups	List with names for groups and names of the group members. If NULL than each member is in the individual group.
.log	if T then log-scale the y axis.

See Also

rarefaction

Examples

```
## Not run:
data(twb)
names(twb)  # "Subj.A" "Subj.B" "Subj.C" "Subj.D"
twb.rar <- rarefaction(twb, .col = "Read.count")
vis.rarefaction(twb.rar, list(A = c("Subj.A", "Subj.B"), B = c("Subj.C", "Subj.D")))

## End(Not run)
```

---

vis.shared.clonotypes *Visualisation of shared clonotypes occurrences among repertoires.*

Description

Visualise counts or proportions of shared clonotypes among repertoires.

Usage

```
vis.shared.clonotypes(.shared.rep, .x.rep = NA, .y.rep = NA,
  .title = "Shared clonotypes", .ncol = 3, .point.size.modif = 1)
```

Arguments

.shared.rep	Shared repertoires, as from shared.repertoire function.
.x.rep	Which repertoire show on x-axis. Either a name or an index of a repertoire in the .shared.rep or NA to choose all repertoires.
.y.rep	Which repertoire show on y-axis. Either a name or an index of a repertoire in the .shared.rep or NA to choose all repertoires.
.title	Main title of the plot.
.ncol	Number of columns in the resulting plot.
.point.size.modif	Modify this to correct sizes of points.

Value

ggplot2 object or plot

See Also

shared.repertoire

Examples

```
## Not run:
data(twb)
# Show shared nucleotide clonotypes of all possible pairs
# using the Read.proportion column
twb.sh <- shared.repertoire(twb, "n0rp")
vis.shared.clonotypes(twb.sh, .ncol = 4)

# Show shared amino acid + Vseg clonotypes of pairs
# including the Subj.A (the first one) using
# the Read.count column.
twb.sh <- shared.repertoire(twb, "avrc")
vis.shared.clonotypes(twb.sh, 1, NA, .ncol = 4)
# same, just another order of axis
vis.shared.clonotypes(twb.sh, NA, 1, .ncol = 4)

# Show shared nucleotide clonotypes of Subj.A (the first one)
# Subj.B (the second one) using the Read.proportion column.
twb.sh <- shared.repertoire(twb, "n0rp")
vis.shared.clonotypes(twb.sh, 1, 2)

# Show the same plot, but with much larget points.
vis.shared.clonotypes(twb.sh, 1, 2, .point.size.modif = 3)

## End(Not run)
```

---

vis.top.proportions *Visualisation of top clones proportions.*

Description

Visualisation of proportion of the top clones.

Usage

```
vis.top.proportions(.data, .head = c(10, 100, 1000, 10000, 30000, 1e+05,
  3e+05, 1e+06), .col = "Read.count")
```

Arguments

.data	Data frame with clones.
.head	Integer vector of clones for the .head parameter for the top.proportion function.
.col	Parameter .col for the top.proportion function.

See Also

top.proportion

Examples

```
## Not run:
vis.top.proportions(immdata)

## End(Not run)
```

Index