Package 'clonotypeR'

October 16, 2019

Type Package

Title High throughput analysis of T cell antigen receptor sequences

Version 1.22.0

Date 2016-10-13

Author Charles Plessy <plessy@riken.jp>

Maintainer Charles Plessy <plessy@riken.jp>

Description High throughput analysis of T cell antigen receptor sequences The genes encoding T cell receptors are created by somatic recombination, generating an immense combination of V, (D) and J segments. Additional processes during the recombination create extra sequence diversity between the V an J segments. Collectively, this hyper-variable region is called the CDR3 loop. The purpose of this package is to process and quantitatively analyse millions of V-CDR3-J combination, called clonotypes, from multiple sequence libraries.

License file LICENSE

Imports methods

Suggests BiocGenerics, edgeR, knitr, pvclust, RUnit, vegan

VignetteBuilder knitr

biocViews Sequencing

URL http://clonotyper.branchable.com/

BugReports http://clonotyper.branchable.com/Bugs/

RoxygenNote 5.0.1

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

git_branch RELEASE_3_9

git_last_commit 03b7c0b

git_last_commit_date 2019-05-02

Date/Publication 2019-10-15

R topics documented:

`clonotypeR` *clonotypeR: High throughput analysis of T cell antigen receptor sequences*

Description

clonotypeR: High throughput analysis of T cell antigen receptor sequences

---

`clonotype_table` *clonotype_table*

Description

Create a table count of clonotypes or other features

Usage

```
clonotype_table(libs, feats = c("V", "pep", "J"), data,
  filter = (data$unproductive | data$ambiguous), minscore = 0,
  minqual = 1, sample = FALSE)
```

Arguments

`libs`	A character vector containing the name of one or many libraries. Same names must not appear twice. If no library names are provided, all the libraries present in the clonotypes data frame will be used.
`feats`	What to count. By default, it counts clonotypes, defined as `c("V","pep","J")`. But it can also count single features, such as the V or J segments.
`data`	Data frame as loaded by `read_clonotypes`.
`filter`	Logical vector to filter out clonotypes. By default it relies on the clonotypes data frame to provide a “unproductive” column that indicates clonotypes with a stop codon or a frame shift, and a “ambiguous” column that indicates clonotypes where the DNA sequences has ambiguous (“N”) nucleotides.
`minscore`	Minimum alignment score. Clonotypes with an alignment score lower than this value are discarded.
`minqual`	Minimum mapping quality. Clonotypes with a mapping quality lower than this value are discarded.
`sample`	Indicate the number of clonotypes to randomly sample from the library (no replacement). Default: no

Details

Using a clonotype data frame loaded with `read_clonotypes`, `clonotype_table` will create a table counting how many times each clonotypes have been seen in each libraries. By default, the unproductive rearrangements are filtered out.

Value

`clonotype_table` returns a data frame, where row names are features (clonotypes, segment names, . . .), column names are libraries, and values are number of times each feature was found in each library.

See Also

`read_clonotypes`

Examples

```
# Read the package's example data
clonotypes <- read_clonotypes(system.file('extdata', 'clonotypes.txt.gz', package = "clonotypeR"))

# Inspect the alignment scores
hist(clonotypes$score)

# Count J segments
j <- clonotype_table(levels(clonotypes$lib), "J", data=clonotypes)

# Normalise counts in parts per million
J <- data.frame(prop.table(as.matrix(j),2) * 1000000)
```

`common_clonotypes` *common_clonotypes*

Description

Reports clonotypes common between libraries.

Usage

```
common_clonotypes(group1, group2, mode, data)

## S4 method for signature 'character,missing,missing,data.frame'
common_clonotypes(group1, data)

## S4 method for signature 'character,character,missing,data.frame'
common_clonotypes(group1,
  group2, data)

## S4 method for signature 'missing,missing,ANY,data.frame'
common_clonotypes(mode = "count",
  data)
```

```
## S4 method for signature 'missing,missing,ANY,matrix'
common_clonotypes(mode, data)
```

Arguments

`group1`	A character vector containing clonotype library names.
`group2`	A character vector containing clonotype library names.
`mode`	Only when producing a matrix of pairwise comparisons: "count" (default) or "abundance", see below.
`data`	A clonotype table where the data is stored.

Details

When given one group of libraries, lists the clonotypes that have been observed at least in one library of that group. The returned list can be used to subset a data frame produced by `clonotype_table`.

When given two groups of libraries, lists the clonotypes that have been observed at least in one library of each group. Groups can contain a single library, in which case the returned list is simply the clonotypes found in both libraries.

When given a table of clonotypes, produces a matrix in which each cell reportsquantitatively the overlap between each pair of libraries.

Value

In "count" mode, each value in a matrix is the number of clonotypes seen in both of the two libraries considered. The matrix is therefore symmetric.

In "abundance" mode, each value indicates, for a given pair of libraries, the cumulative abundance of the common clonotypes (seen in both libraries), calculated for the library indicated by the row. The matrix is therefore not symmetric. For instance, a pair of libraries A and B can have 100 sequences each in total, one clonotype in common, which is found 8 times in A, but 54 times in B.

Methods (by class)

- `group1 = character,group2 = missing,mode = missing,data = data.frame`: Reports clonotypes common between libraries.
- `group1 = character,group2 = character,mode = missing,data = data.frame`: Reports clonotypes common between libraries.
- `group1 = missing,group2 = missing,mode = ANY,data = data.frame`: Reports clonotypes common between libraries.
- `group1 = missing,group2 = missing,mode = ANY,data = matrix`: Reports clonotypes common between libraries.

See Also

`clonotype_table`, `unique_clonotypes`

Examples

```
# Load example data
clonotypes.long <- read_clonotypes(system.file('extdata', 'clonotypes.txt.gz', package = "clonotypeR"))
clonotypes <- clonotype_table(levels(clonotypes.long$lib), data=clonotypes.long)
summary(clonotypes)

# List clonotypes found in library A, and B or C.
common_clonotypes(group1="A", group2=c("B","C"), data=clonotypes)

# Count clonotypes found in library A, and B or C.
length(common_clonotypes(group1="A", group2=c("B","C"), data=clonotypes))

# Matrix of numbers of common clonotypes
common_clonotypes(data=clonotypes)

# Matrix of abundance of common clonotypes
common_clonotypes(data=clonotypes, mode="abundance")
```

`extdata` *Extra data used to calculate ID numbers in Yassai et al.'s nomenclature.*

Description

Data frame derived from Table 1 of Yassai et al., 2009, to construct clonotype names.

Details

`V_after_C`: sequence of the V segments after their conserved cystein.

`J_before_FGxG`: sequence of the J segments before their conserved FGxG motif.

`codon_ids`: data frame derived from Table 1 of Yassai et al., 2009, to construct clonotype names.

The V_after_C and J_before_FGxG tables are, generated from the mouse reference data with the command: `make refresh-data` in the source repository of clonotypeR.

Value

codon_ids:

`codon`	Nucleotide triplet.
`aminoacid`	Single-letter amino acid abbreviation ("O" for stop).
`id`	ID numbers assigned to the codons for each amino acids.

J_before_FGxG:

`row name`	J segment name, for instance "TRAJ61".
`sequence`	Sequence of the nucleic acids preceding the first codon of the conserved FGxG motif.

V_after_C:

`row name`	V segment name, for instance "TRAV1".
`sequence`	Sequence of the nucleic acids following the codon of the conserved cysteine.

References

A clonotype nomenclature for T cell receptors. Maryam B. Yassai, Yuri N. Naumov, Elena N. Naumova and Jack Gorski Immunogenetics, 2009, Volume 61, Number 7, Pages 493-502

See Also

`yassai_identifier`

Examples

```
V_after_C <- read.table(system.file('extdata', 'V_after_C.txt.gz', package = "clonotypeR"), stringsAsFactors=

J_before_FGxG <- read.table(system.file('extdata', 'J_before_FGxG.txt.gz', package = "clonotypeR"), stringsAs

codon_ids <- read.table(system.file('extdata', 'codon_ids.txt.gz', package = "clonotypeR"), header=TRUE, row.
```

`is_unproductive` *is_unproductive*

Description

Determines if clonotype sequences are productive.

Usage

```
is_unproductive(data)
```

Arguments

`data`	Data frame of clonotype sequences, or character vector describing a single clonotype, where the DNA sequence is available under the name "dna" and its translation available under the name "pep". Clonotypes are marked unproductive if the length of their DNA sequence is not a multiple of 3, or if they contain a stop codon, as indicated by an asterisk in the translated sequence.

Details

ClonotypeR identifies V and J segments, isolates the DNA sequence between the conserved cystein and the FGxG motifs, and translates it. This functions verifies that this sequence is in frame and has no stop codon.

Value

Logical vector, with one value per row in the original data.

See Also

`read_clonotypes`

Examples

```
clonotypes <- read_clonotypes(system.file('extdata', 'clonotypes.txt.gz', package = "clonotypeR"))
is_unproductive(clonotypes)
```

`private_clonotypes` *private_clonotypes*

Description

List clonotypes found exclusively in one library.

Usage

```
private_clonotypes(..., data)
```

Arguments

`...`	Library names.
`data`	A clonotype table.

Value

A vector of clonotype names.

See Also

`clonotype_table`

Examples

```
clonotypes <- read_clonotypes(system.file('extdata', 'clonotypes.txt.gz', package = "clonotypeR"))
clonotypes <- clonotype_table(levels(clonotypes$lib), data=clonotypes)
private_clonotypes("C", data=clonotypes)
```

`read_clonotypes` *read_clonotypes*

Description

Reads a clonotype_table and returns a data frame.

Usage

```
read_clonotypes(filename, scores = TRUE, ...)
```

Arguments

`filename`	Path to the tabulation-delimited text file containing the extracted clonotypes.
`scores`	Set to false to load legacy data that did not contain “score” and “mapq” columns.
`...`	The rest of the arguments are passed to the `read.table()` function.

Details

Reads a clonotype_table in a TAB-separated or OSCT format, and returns a data frame that has eight columns, for library name, V and J segments names, sequence read identifier, DNA, sequence quality, aminoacid sequence of the CDR3 region, mark for unproductive recombinations, and mark for ambiguous sequences.

Value

`lib`	Library name (factor).
`V`	V segment name (factor).
`J`	J segment name (factor).
`score`	Alignment score (numeric).
`mapq`	Mapping quality (numeric). A sequence with a good alignment score will still have a low mapping quality if there are good alternative alignments to other V segments.
`read`	Sequence read identifier (character).
`dna`	DNA sequence of the CDR3 region (character).
`qual`	Quality values for the DNA sequence (character).
`pep`	Translation of the DNA sequence (character).
`unproductive`	Flag indicating stop codons or frame shifts (logical).
`ambiguous`	Flag indicating that the DNA sequences has ambiguous (“N”) nucleotides (logical).

See Also

`clonotype_table`, `is_unproductive`, `read.table`, Order Switchable Column Table (OSCT, `http://sourceforge.net/projects/osctf/`)

Examples

```
clonotypes <- read_clonotypes(system.file( 'extdata', 'clonotypes.txt.gz'
                                         , package = "clonotypeR"))
```

`unique_clonotypes` *unique_clonotypes*

Description

Lists unique clonotypes in libraries

Usage

```
unique_clonotypes(..., data)
```

Arguments

`...`	One or more character vectors contain clonotype library names.
`data`	The name of the clonotype_table where the data is stored.

Details

Finds all the clonotypes expressed in one or more libraries, and returns a vector where they are listed once. This vector can be used to subset a clonotype_table.

Value

Character vector of clonotype names. Their order follows the original row name order of the clonotype_table.

See Also

`clonotype_table`, `common_clonotypes`

Examples

```
# Load example data
clonotypes.long <- read_clonotypes(system.file('extdata', 'clonotypes.txt.gz', package = "clonotypeR"))
clonotypes <- clonotype_table(levels(clonotypes.long$lib), data=clonotypes.long)
summary(clonotypes)

# List clonotypes found in library A.
unique_clonotypes("A", data=clonotypes)

# List clonotypes found in library A or B.
unique_clonotypes("A","B", data=clonotypes)
```

yassai_identifier *yassai_identifier*

Description

TCR clonotype identifier (Yassai et al.)

Usage

```
yassai_identifier(data, V_after_C, J_before_FGxG, long = FALSE)

## S4 method for signature 'character,data.frame,data.frame,ANY'
yassai_identifier(data,
  V_after_C, J_before_FGxG, long = FALSE)

## S4 method for signature 'ANY,missing,missing,ANY'
yassai_identifier(data, long)

## S4 method for signature 'data.frame,data.frame,data.frame,logical'
yassai_identifier(data,
  V_after_C, J_before_FGxG, long = FALSE)
```

Arguments

`data`	A data frame or a character vector containing a clonotype(s) with proper row or element names.
`V_after_C`	(optional) A data frame indicating the aminoacids following the conserved cystein for each V segment.
`J_before_FGxG`	(optional) A data frame indicating the aminoacids preceding the conserved FGxG motif for each V segment.
`long`	(optional) Avoids identifier collisions by displaying the codons, and indicating the position of the V–J junction in ambiguous cases.

Details

The clonotype nomenclature defined by Yassai et al. in `http://dx.doi.org/10.1007/s00251-009-0383-x`.

By default, `yassai_identifier()` assume mouse sequences and will load the V_after_C and J_before_FGxG tables distributed in this package. It is possible to provide alternative tables either by passing them directly as argument, or by installing them as “./inst/extdata/V_after_C.txt.gz” and “./inst/extdata/J_before_FGxG.txt.gz”.

Some clonotypes have a different DNA sequence but the same identifier following the original nomenclature (see below for examples). The ‘long’ mode was created to avoid these collisions. First, it displays all codons, instead of only the non-templated ones and their immediate neighbors. Second, for the clonotypes where all codons are identical to the V or J germline sequence, it indicates the position of the V–J junction in place of the codon IDs.

Value

The name (for instance sIRSSy.1456B19S1B27L11) consists of five segments:

1. CDR3 amino acid identifier (ex. sIRSSy), followed by a dot;
2. CDR3 nucleotide sequence identifier (ex. 1456);
3. variable (V) segment identifier (ex. BV19S1);
4. joining (J) segment identifier (ex. BJ2S7);
5. CDR3 length identifier (ex. L11).

Methods (by class)

- `data = character,V_after_C = data.frame,J_before_FGxG = data.frame,long = ANY`: TCR clonotype identifier (Yassai et al.)
- `data = ANY,V_after_C = missing,J_before_FGxG = missing,long = ANY`: TCR clonotype identifier (Yassai et al.)
- `data = data.frame,V_after_C = data.frame,J_before_FGxG = data.frame,long = logical`: TCR clonotype identifier (Yassai et al.)

See Also

`codon_ids`, `J_before_FGxG`, `V_after_C`

Examples

```
clonotypes <- read_clonotypes(system.file('extdata', 'clonotypes.txt.gz', package = "clonotypeR"))
head(yassai_identifier(clonotypes))

# The following two clonotypes have a the same identifier, and are
# disambiguated by using the long mode

yassai_identifier(c(V="TRAV14-1", J="TRAJ43", dna="GCAGCTAATAACAACAATGCCCCACGA", pep="AANNNNAPR"))
# [1] "aAn.1A14-1A43L9"

yassai_identifier(c(V="TRAV14-1", J="TRAJ43", dna="GCAGCAGCTAACAACAATGCCCCACGA", pep="AAANNNAPR"))
# [1] "aAn.1A14-1A43L9"

yassai_identifier(c(V="TRAV14-1", J="TRAJ43", dna="GCAGCTAATAACAACAATGCCCCACGA", pep="AANNNNAPR"), long=TRUE
# [1] "aAnnnnapr.1A14-1A43L9"

yassai_identifier(c(V="TRAV14-1", J="TRAJ43", dna="GCAGCAGCTAACAACAATGCCCCACGA", pep="AAANNNAPR"), long=TRUE
# [1] "aaAnnnapr.1A14-1A43L9"

# The following two clonotypes would have the same identifier in long mode
# if the position of the V-J junction would not be indicated in place of the
# codon IDs.

yassai_identifier(c(V="TRAV14N-1", J="TRAJ56", dna="GCAGCTACTGGAGGCAATAATAAGCTGACT", pep="AATGGNNKLT"), long
# [1] "aatggnnklt.1A14N1A56L10"

yassai_identifier(c(V="TRAV14N-1", J="TRAJ56", dna="GCAGCAACTGGAGGCAATAATAAGCTGACT", pep="AATGGNNKLT"), long
# [1] "aatggnnklt.2A14N1A56L10"
```

Index