

# Package ‘clonotypeR’

June 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 and 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** 2019-06-15

## R topics documented:

clonotypeR . . . . .	2
clonotype_table . . . . .	2
common_clonotypes . . . . .	3
extdata . . . . .	5

is_unproductive . . . . .	6
private_clonotypes . . . . .	7
read_clonotypes . . . . .	7
unique_clonotypes . . . . .	9
yassai_identifier . . . . .	10

## Index 12

---

clonotypeR	<i>clonotypeR: High throughput analysis of T cell antigen receptor sequences</i>
------------	--

---

### Description

clonotypeR: High throughput analysis of T cell antigen receptor sequences

---

clonotype_table	<i>clonotype_table</i>
-----------------	------------------------

---

### 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 <a href="#">read_clonotypes</a> .
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	<i>common_clonotypes</i>
-------------------	--------------------------

---

## 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 reports quantitatively 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	<i>Extra data used to calculate ID numbers in Yassai et al.'s nomenclature.</i>
---------	---

---

**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	<i>is_unproductive</i>
-----------------	------------------------

---

## 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 cysteine 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, ...)
```





---

unique_clonotypes	<i>unique_clonotypes</i>
-------------------	--------------------------

---

### 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

<code>data</code>	A data frame or a character vector containing a clonotype(s) with proper row or element names.
<code>V_after_C</code>	(optional) A data frame indicating the aminoacids following the conserved cystein for each V segment.
<code>J_before_FGxG</code>	(optional) A data frame indicating the aminoacids preceding the conserved FGxG motif for each V segment.
<code>long</code>	(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="AANNNNAPR"))
# [1] "aAn.1A14-1A43L9"
```

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

```
yassai_identifier(c(V="TRAV14-1", J="TRAJ43", dna="GCAGCAGCTAACAACAATGCCCCACGA", pep="AANNNNAPR"), 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=TRUE)
# [1] "aatggnnkl.1A14N1A56L10"
```

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

# Index

## \*Topic **Yassai**

extdata, [5](#)

## \*Topic **clonotype**

extdata, [5](#)

## \*Topic **nomenclature**

extdata, [5](#)

clonotype\_table, [2](#), [3](#), [4](#), [7–9](#)

clonotypeR, [2](#)

clonotypeR-package (clonotypeR), [2](#)

codon\_ids, [11](#)

codon\_ids (extdata), [5](#)

common\_clonotypes, [3](#), [9](#)

common\_clonotypes, character, character, missing, data.frame-method  
(common\_clonotypes), [3](#)

common\_clonotypes, character, missing, missing, data.frame-method  
(common\_clonotypes), [3](#)

common\_clonotypes, missing, missing, ANY, data.frame-method  
(common\_clonotypes), [3](#)

common\_clonotypes, missing, missing, ANY, matrix-method  
(common\_clonotypes), [3](#)

extdata, [5](#)

is\_unproductive, [6](#), [8](#)

J\_before\_FGxG, [11](#)

J\_before\_FGxG (extdata), [5](#)

private\_clonotypes, [7](#)

read.table, [8](#)

read\_clonotypes, [2](#), [3](#), [6](#), [7](#)

unique\_clonotypes, [4](#), [9](#)

V\_after\_C, [11](#)

V\_after\_C (extdata), [5](#)

yassai\_identifier, [6](#), [10](#)

yassai\_identifier, ANY, missing, missing, ANY-method  
(yassai\_identifier), [10](#)

yassai\_identifier, character, data.frame, data.frame, ANY-method  
(yassai\_identifier), [10](#)

yassai\_identifier, data.frame, data.frame, data.frame, logical-method  
(yassai\_identifier), [10](#)