tcR: a package for T-cell receptor repertoire data analysis

Vadim Nazarov
vdm.nazarov@gmail.com

Mikhail Pogorelyy
m.pogorely@gmail.com

August 2014

Abstract

Abstract?

Contents

1 Introduction

1.1 Overview and vignette structure

The *tcR* package is designed to represent sequences, ranges representing indices along those sequences, and data related to those ranges. In this vignette, we will rely on simple, illustrative example datasets, rather than large, real-world data, so that each data structure and algorithm can be explained in an intuitive, graphical manner. We expect that packages that apply to a particular problem domain will provide vignettes with relevant, realistic examples. In Section 1 we introduce the tcR package,

1.2 Example pipeline

You can find the pipeline of the twins analysis here:

```
<path to the tcR package>/inst/twins.pipeline.Rmd
```

In RStudio you can run it as follows:

```
Run RStudio -> load the pipeline .Rmd files -> press the knitr button
```

1.3 MiTCR: a tool for retrieving CDR3 sequences from NGS data

how to use mitcr

```
java -Xmx8g -jar mitcr.jar -pset flex -level 2 ~/data/raw/TwA1_B.fastq.gz ~/data/mitcr/TwA1_B.txt
```

You can start MiTCR from an R session with `start.mitcr` function. E.g., to run code above you need to do following:

```
> start.mitcr('raw/TwA1_B.fastq.gz', 'mitcr/TwA1_B.txt', .file.path = '~/data/',
+             pset = 'flex', level = 1, 'debug', .mitcr.path = '~/programs/', .mem = '8g')
```

Run MiTCR on all files from the ' /data/raw/' directory:

```
> start.mitcr(.file.path = '~/data/raw', pset = 'flex', level = 1, 'debug',
+             .mitcr.path = '~/programs/', .mem = '8g')
```

1.4 Structure of a MiTCR data frame (clonesets representation)

Package basically operates with data frames with specific column names, which called MiTCR data frames.

Dataflow is next:

```
NGS .fastq files -> run MiTCR -> tab-separated files with data -> tcR parser
```

Barcodes parse.file parse.folder parse.file.list

In our analysis only few columns are broadly used. Hence, to do almost all analysis you just need a data frames with following columns:

- *Read.count*
- *CDR3.amino.acid.sequence*
- *V.segments*

Additionally, for analysis of J-segments usage or nucleotide sequences intersection (see Section 4.) you should provide:

- *J.segments*

- *CDR3.nucleotide.sequence*

Any data frame with this columns will be suitable for using with the package.

2 Repertoire descriptive statistics

To describe a repertoire, we need to compute a number of informative statistics.

2.1 Sequences summary

To get a general view of CDR3 sequences counts (overall count of sequences, in- and out-of-frames numbers and percentage) use the `mitcr.stats` function. It returns a `summary` of counts of nucleotide sequences ('clones') and aminoacid sequences ('clonotypes'), as well as summary of read counts:

```
> library(tcR)
> mitcr.stats(immdata)
```

```
                          TwA1_B       TwA2_B       TwC1_B       TwC2_B
#Nucleotide clones    1.162990e+05 1.038670e+05 1.606440e+05 1.886470e+05
#Aminoacid clonotypes 5.703500e+04 9.968000e+04 1.540290e+05 1.816620e+05
%Aminoacid clonotypes 4.904169e-01 9.596888e-01 9.588220e-01 9.629732e-01
#In-frames            1.122190e+05 9.813500e+04 1.543670e+05 1.734600e+05
%In-frames            9.649180e-01 9.448140e-01 9.609260e-01 9.194951e-01
#Out-of-frames        4.080000e+03 5.732000e+03 6.277000e+03 1.518700e+04
%Out-of-frames        3.508199e-02 5.518596e-02 3.907398e-02 8.050486e-02
Sum.Read.count        3.987648e+06 2.769645e+06 2.124464e+06 3.273701e+06
Min.Read.count        8.000000e+00 1.000000e+00 4.000000e+00 4.000000e+00
1st Qu.Read.count     1.000000e+01 2.000000e+00 5.000000e+00 6.000000e+00
Median.Read.count     1.200000e+01 5.000000e+00 7.000000e+00 9.000000e+00
Mean.Read.count       3.429000e+01 2.667000e+01 1.322000e+01 1.735000e+01
3rd Qu.Read.count     1.800000e+01 1.000000e+01 1.000000e+01 1.400000e+01
Max.Read.count        8.152000e+04 1.712000e+05 1.046000e+05 3.359000e+04
                          TwD1_B       TwD2_B
#Nucleotide clones    1.754220e+05 1.509980e+05
#Aminoacid clonotypes 1.681460e+05 1.444310e+05
%Aminoacid clonotypes 9.585229e-01 9.565094e-01
#In-frames            1.703490e+05 1.471520e+05
%In-frames            9.710812e-01 9.745295e-01
#Out-of-frames        5.073000e+03 3.846000e+03
%Out-of-frames        2.891884e-02 2.547054e-02
Sum.Read.count        1.976347e+06 2.339886e+06
Min.Read.count        3.000000e+00 4.000000e+00
1st Qu.Read.count     4.000000e+00 5.000000e+00
Median.Read.count     6.000000e+00 7.000000e+00
Mean.Read.count       1.127000e+01 1.550000e+01
3rd Qu.Read.count     9.000000e+00 1.000000e+01
Max.Read.count        4.471000e+04 1.782000e+05
```

2.2 Percentage and counts of the most abundant clonotypes

Function `clonal.proportion` is used to get the number of most abundant by the count of reads clones. E.g., compute number of clones which fill up approximately the 25% of the sum of values in "Read.count":

```
>                                # How many clones fill up approximately
> clonal.proportion(immdata, 25)  # the 25% of the sum of values in 'Read.count'?
```

```
           TwA1_B TwA2_B TwC1_B TwC2_B TwD1_B TwD2_B
Clones       53.0    8.0    788    622   1146     65
Percentage   25.1   26.2     25     25     25     25
```

To get a proportion of sum of reads of top clones to the overall sum of reads, use `head.proportion`, i.e. get $(\sum$ `reads of top clones`$)/(\sum$ `reads for all clones`$)$. E.g., get a proportion of the top-10 clones' reads to the overall number of reads:

```
>                                # What proportion of the top-10 clones' reads
> head.proportion(immdata, 10)  # to the overall number of reads?
```

```
    TwA1_B     TwA2_B     TwC1_B     TwC2_B     TwD1_B     TwD2_B
0.10925087 0.29659794 0.11962641 0.05658825 0.07898967 0.20067986
```

Function `split.proporion` with two arguments `.col` and `.bound` gets subset of the given data frame with reads, which have column's `.col` value $\leq$ `.bound` and computes the ratio of sums of count reads of such subset to the overall data frame. E.g., get proportion of sum of reads of sequences which has "Read.count" <= 100 to the overall number of reads:

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

```
   TwA1_B    TwA2_B    TwC1_B    TwC2_B    TwD1_B    TwD2_B
0.9768012 0.9869160 0.9910050 0.9895042 0.9937294 0.9903575
```

2.3 In- and out-of-frame CDR3 sequences subsetting and statistics

Functions for performing subsetting and counting cardinality of in-frame and out-of-frame subsets are: `count.*frames`, `get.*frames`. Parameter `.head` for this functions is a parameter to the `head` function, that applied before subsetting. Functions accept both data frames and list of data frames as parameters. E.g., get data frame with only in-frame sequences and count out-of-frame sequences in the first 5000 rows for this data frame:

```
> imm.in <- get.inframes(immdata) # Return all in-frame sequences from the 'immdata'.
>                                 # Count the number of out-of-frame sequences
> count.outframes(immdata, 5000)  # from the first 5000 sequences.
```

```
TwA1_B TwA2_B TwC1_B TwC2_B TwD1_B TwD2_B
   184    212     73    326    108     82
```

General function with parameter stands for 'all' (all sequences), 'in' (only in-frame sequences) or 'out' (only out-of-frame sequences) is `count.frames`:

```
> imm.in <- get.frames(immdata, 'in') # Similar to 'get.inframes(twb)'.
> count.frames(immdata[[1]], 'all')   # Just return number of rows.
```

```
[1] 116299
```

```
> flag <- 'out'
> count.frames(immdata, flag, 5000)   # Similar to 'count.outframes(twb, 5000)'.
```

```
TwA1_B TwA2_B TwC1_B TwC2_B TwD1_B TwD2_B
   184    212     73    326    108     82
```

2.4 Segments statistics

To access V- and J-usage of a repertoire, *tcR* provides functions `freq.segments`, `freq.segments.2D` and a family of functions `freq.[VJ][ab]` for simplier use. Function `freq.segments`, depending on parameters, computes frequencies or counts of the given elements (e.g., V-segments) in the given column (e.g., "V.segments") of the input data frame(s). Function `freq.segments.2D` computes joint distributions or counts of the two given elements (e.g., V-segments and J-segments). For plotting V-usage and J-usage see section about plots. V and J alphabets are store in the .rda file "human.alphabets.rda". All of mentioned functions are accepts data frames as well as list of data frames. Output for this functions are data frames with the first column stands for segment and other or others for frequencies.

```
> data(human.alphabets)
> imm1.vs <- freq.segments(immdata[[1]])  # Equivalent to freq.Vb(immdata[[1]])
> head(imm1.vs)
```

```
       Segment        Freq
Other    Other 0.001572868
1     TRBV10-1 0.003068381
2     TRBV10-2 0.005354626
3     TRBV10-3 0.031551896
4     TRBV11-1 0.005234297
5     TRBV11-2 0.019381511
```

```
> imm.vs.all <- freq.segments(immdata)  # Equivalent to freq.Vb(immdata)
> imm.vs.all[1:10, 1:4]
```

```
              Segment      TwA1_B      TwA2_B      TwC1_B
1               Other 0.001572868 0.007727395 0.001991375
2            TRBV10-1 0.003068381 0.003281497 0.002034936
3            TRBV10-2 0.005354626 0.004801955 0.002501665
4            TRBV10-3 0.031551896 0.021450017 0.039821274
5            TRBV11-1 0.005234297 0.004532507 0.003553360
6            TRBV11-2 0.019381511 0.025520613 0.018308203
7            TRBV11-3 0.003051191 0.004513261 0.003802281
8  TRBV12-4, TRBV12-3 0.052385946 0.049058855 0.069679451
9            TRBV12-5 0.002569877 0.002415412 0.004928653
10             TRBV13 0.007279884 0.004311174 0.005264697
```

```
> imm1.vj <- freq.segments.2D(immdata[[1]])
> imm1.vj[1:5, 1:5]
```

```
   Segment      TRBJ1-1      TRBJ1-2      TRBJ1-3      TRBJ1-4
1 TRBV10-1 0.0004711365 7.709506e-05 7.709506e-05 9.422729e-05
2 TRBV10-2 0.0007109878 3.854753e-04 9.422729e-05 1.970207e-04
3 TRBV10-3 0.0041031703 2.544137e-03 8.480456e-04 1.002236e-03
4 TRBV11-1 0.0010365002 1.284918e-04 4.283059e-05 5.996282e-05
5 TRBV11-2 0.0022357567 1.207823e-03 5.396654e-04 8.994423e-04
```

2.5 Search for a target CDR3 sequences

For exact or fuzzy search of sequences the package employed the function `find.clonotypes`. Input arguments for this function are data frame or list of data frames, targets (character vector or data frame with column for sequences and additional columns like V-segments), value of which column or columns return, method which will be used in comparison of sequences among each other (either "exact" for exact matching, "hamm" for matching sequences by Hamming distance (two sequences are matched if H <= 1) or "lev" for matching sequences by Levenshtein distance (two sequences are matched if L <= 1)), and columns name for getting sequences for matching from the given data. Sounds very complex, but in practice it's very easy, therefore let's go to examples. Suppose we want to search for a some CDR3 sequences in a number of repertoires:

```
> cmv
```

```
  CDR3.amino.acid.sequence V.segments
1              CASSSANYGYTF    TRBV4-1
2              CSVGRAQNEQFF    TRBV4-1
3             CASSLTGNTEAFF    TRBV4-1
4          CASSALGGAGTGELFF    TRBV4-1
5          CASSLIGVSSYNEQFF    TRBV4-1
```

We will search for them using all methods of matching (exact, hamming or levenshtein) and with and without using V-segments. Also, for the first case (exact matching and without V-segment) we return "Total.insertions"

column along with the "Read.count" column, and for the second case output will be a "Rank" - rank of a clone or a clonotype in a data frame.

```
> cmv.imm.ex <- find.clonotypes(immdata[1:2], cmv[,1], 'exact',
+                               c('Read.count', 'Total.insertions'), .verbose = F)
> head(cmv.imm.ex)
```

```
                  CDR3.amino.acid.sequence Read.count.TwA1_B Read.count.TwA2_B
CASSALGGAGTGELFF           CASSALGGAGTGELFF               153               319
CASSALGGAGTGELFF.1         CASSALGGAGTGELFF               153                35
CASSLTGNTEAFF                 CASSLTGNTEAFF                35               263
CASSLTGNTEAFF.1               CASSLTGNTEAFF                35                35
CASSLTGNTEAFF.2               CASSLTGNTEAFF                35                28
CASSLTGNTEAFF.3               CASSLTGNTEAFF                35                 1
                  Total.insertions.TwA1_B Total.insertions.TwA2_B
CASSALGGAGTGELFF                       9                      10
CASSALGGAGTGELFF.1                     9                       9
CASSLTGNTEAFF                          2                       2
CASSLTGNTEAFF.1                        2                       0
CASSLTGNTEAFF.2                        1                       1
CASSLTGNTEAFF.3                        1                       4
```

```
> cmv.imm.hamm.v <- find.clonotypes(immdata[1:3], cmv, 'hamm','Rank',
+                                  .target.col = c('CDR3.amino.acid.sequence', 'V.segments'), .verbose = F)
> head(cmv.imm.hamm.v)
```

```
                 CDR3.amino.acid.sequence V.segments Rank.TwA1_B Rank.TwA2_B
CAAAPTNTGELFF                CAAAPTNTGELFF    TRBV4-1          NA          NA
CAAGDNNSPLHF                  CAAGDNNSPLHF    TRBV4-1          NA          NA
CAAGRGGTYNEQFF               CAAGRGGTYNEQFF    TRBV4-1          NA          NA
CACSLSQDRSFPDF               CACSLSQDRSFPDF    TRBV4-1          NA       80420
CACSQRRDRARVEKLFF        CACSQRRDRARVEKLFF    TRBV4-1          NA       62723
CAEQP*GQ~PRETQYF          CAEQP*GQ~PRETQYF    TRBV4-1          NA       75229
                 TwC1_B.Rank
CAAAPTNTGELFF          79991
CAAGDNNSPLHF           41609
CAAGRGGTYNEQFF         99191
CACSLSQDRSFPDF            NA
CACSQRRDRARVEKLFF         NA
CAEQP*GQ~PRETQYF          NA
```

```
> cmv.imm.lev.v <- find.clonotypes(immdata[1:3], cmv, 'lev',
+                                  .target.col = c('CDR3.amino.acid.sequence', 'V.segments'), .verbose = F)
> head(cmv.imm.lev.v)
```

```
                CDR3.amino.acid.sequence V.segments Read.count.TwA1_B
CASSALGGAGTGELFF         CASSALGGAGTGELFF    TRBV4-1               NA
CASSELTGNTEAFF             CASSELTGNTEAFF    TRBV4-1               13
CASSELTGNTEAFF.1           CASSELTGNTEAFF    TRBV4-1               13
CASSLGGNTEAFF               CASSLGGNTEAFF    TRBV4-1               NA
CASSLIGVSSYNEQFF         CASSLIGVSSYNEQFF    TRBV4-1               NA
CASSLRTGNTEAFF             CASSLRTGNTEAFF    TRBV4-1               NA
                Read.count.TwA2_B TwC1_B.Read.count
CASSALGGAGTGELFF                NA                NA
CASSELTGNTEAFF                  NA                NA
CASSELTGNTEAFF.1                NA                NA
CASSLGGNTEAFF                   NA                 5
CASSLIGVSSYNEQFF                NA                NA
CASSLRTGNTEAFF                  NA                 9
```

3 Analysis of sets and distributions of receptors

Repertoires (both TCRs and BCRs) can be viewed as sets of elements, e.g. sets of CDR3 amino acid sequences of sets of tuples (CDR3 amino acid sequence, V-segment). *tcR* provides functions for evaluating similarity and diversity of such sets.

3.1 Intersections between sets of CDR3 sequences

concept with / without normalisation aminoacid / nucleotide / column straight / hamming / levenshtein without segments / wuth V-segments / with V+J-segments

```
> cross(immdata[[1]], immdata[[2]], 'n0c')  # Equivalent to intersect(immdata[[1]]$CDR3.nucleotide.sequence,
[1] 1408
>                                           #                        immdata[[1]]$CDR3.nucleotide.sequence)
```

3.2 Top cross

concept and algorithm plot

3.3 Diversity and skewness

diversity inverse.simpson gini See also the `entropy` function for accessing the repertoire diversity, which is described at the next Section.

3.4 More complicated set similarity measures

cosine.similarity tversky.index overlap.coef morisitas.index

4 Analysis of segments usage

Segments usage statistics could be seen as a discrete distribution. To evaluate how two distributions are close to each other we use measures from the information theory and PCA.

4.1 Information measures

entropy js.div js.div.seg

4.2 PCA

pca.segments pca.segments.2D

5 Shared repertoire of sequences

... E.g., they can be TCRs to a common virus in an environment, indicate similarity of thymus selection in two subjects, or be a most frequent TCRs due to the assembling probability. To support investigations of shared repertoires' properties we provide next functions. Function `shared.repertoire` generates a repertoire of shared TCRs' clone or clonotypes (with or without checking V-segments of similar sequences). As an example, let's generate a shared repertoire of clonotypes with V-segments check and filter out sequences which exists only at one repertoire:

```
> twb.shared <- shared.repertoire(twb, .type = 'av', .min.ppl = 2)
```

For each data frame there are a *Read.count* value of related pair "sequence + V-segment" (you can change the target column(s) by providing the column(s) name(s) to the function). Run Principal Component Analysis (PCA) on *Read.count* values with parameter `scale.` to `prcomp` function and plot a result with functon `shared.seq.pca` (see Section 7.5 "PCA" for another way to plot PCA result):

```
> shared.seq.pca(twb.shared, scale. = T, .name = 'Shared repertoire (min:2, max:6)')
```

Twins pairs are close to each other, due to the high number of shared CDR3 sequences among the most abundant CDR3 sequences. (see Section 4.2 "Top cross"). To get a statistics how many shared CDR3s in how many subjects are presented, use `shared.representation` function:

```
> shared.representation(twb.shared)
```

6 Plots

6.1 Grid plot

`plot.grid.stats`

6.2 Massive clones proportion

6.3 Segments usage

`plot.V.usage` `plot.J.usage`

6.4 Spectratyping

`spectratyping`

6.5 PCA

`plot.pca`

```
> pca.df <- shared.seq.pca(twb.shared, scale. = T, .plot = F)
> plot.pca(pca.df, list(A = c(1,2), B = c(3,4), C = c(5,6)))
> # pca.res <- kmers(...)
> # plot.pca(pca.res)
```

6.6 Venn diagrams for shared repertoire

`plot.shared.venn`

7 Data filters

`mitcr.filter` `count.similar.from.bottom` `remove.similar.from.bottom`

8 Conclusion

Feel free to contact us for package-related or computational immunology research-related questions.

9 Appendix A: Kmers retrieving

10 Appendix B: Nucleotide and aminoacid sequences manipulation

The *tcR* also provides a few number of quick functions for performing classic bioinformatics tasks on strings.

10.1 Nucleotide sequence manipulation

Functions for basic nucleotide sequences manipulations: reverse-complement, translation and GC-content computation. All functions are vectorised.

```
> rev.comp(twb[[1]]$CDR3.nucleotide.sequence)
> bunch.translate(twb[[1]]$CDR3.nucleotide.sequence)
> gc.content(twb[[1]]$CDR3.nucleotide.sequence)
```

10.2 Translation subroutines

codon.variants translated.nucl.sequences reverse.translation

10.3 Find similar sequences by Levenshtein distance

count.neighbors generate.neighbors levenshtein.search

```
> twb.1 <- twb[[1]]
> count.neighbors('CASSLGIHYEQYF')        # 580
> generate.neighbors('CASSLGIHYEQYF')     # Return sequences like 'AASSLGIHYEQYF', 'DASSLGIHYEQYF', ...
> levenshtein.search(get.inframes(twb.1), # Find indices of proximal out-frames in in-frames.
+                    get.outframes(twb.1))
```

11 Appendix C: Twins data description

References

[1] MiTCR: the paper