Tool for Immunoglobulin Genotype Elucidation via Rep-Seq (TIgGER)

Daniel Gadala-Maria

Last modified 2015-04-29

Introduction

Immunoglobulin Repertoire-Sequencing (Rep-Seq) data is currently the subject of much study. A key step in analyzing these data involves assigning the closest known V(D)J germline alleles to the (often somatically mutated) sample sequences using a tool such as IMGT/HighV-QUEST[1]. However, if the sample utilizes alleles not in the germline database used for alignment, this step will fail. Additionally, this alignment has an associated error rate of ~5%[2], notably among sequences carrying a large number of somatic mutations.

Here we provide a **T**ool for **I**mmuno**g**lobulin **G**enotype **E**lucidation via **R**ep-Seq (TIgGER). TIgGER addresses these issues by inferring the set of Ig alleles carried by an individual (including any novel alleles) and then using this set of alleles to correct the initial assignments given to sample sequences by existing tools.

Additional information is available at http://clip.med.yale.edu/tigger/ and in:

Gadala-Maria D, Yaari G, Uduman M, Kleinstein SH (2015) Automated analysis of high-throughput B cell sequencing data reveals a high frequency of novel immunoglobulin V gene segment alleles. *PNAS* 112(8):E862-70.

Input

TIgGER requires two main inputs:

1. Pre-processed Rep-Seq data
2. Database germline sequences

Rep-seq data is input as a data frame where each row represents a unique observation and and columns represent data about that observation. The preferred names of the required columns are provided below along with a description of each.

Column Name	Description
`SEQUENCE_IMGT`	V(D)J sequence gapped in the IMGT gapped format[3]
`V_CALL`	(Comma separated) name(s) of the nearest V allele(s)
`J_CALL`	(Comma separated) name(s) of the nearest J allele(s)
`JUNCTION_LENGTH`	Length of the junction region of the V(D)J sample

An example dataset is provided with the `tigger` package. It contains unique functional sequences assigned to IGHV1 family genes isolated from individual PGP1 (referenced in Gadala-Maria *et al.*).

```
library(tigger)
```

```
## Loading required package: dplyr
##
## Attaching package: 'dplyr'
##
## The following object is masked from 'package:stats':
##
##     filter
##
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
# Load example Rep-Seq data
data(sample_db)
```

The database of germline sequences should be provided in FASTA format with sequences gapped according to the IMGT numbering scheme[3]. IGHV alleles in the IMGT database (build 201408-4) are provided with this package. You may read in your own fasta file using `readGermlineDb()`.

```
# Load example germline database
data(germline_ighv)
```

Running TIgGER

The function most users will wish to use is `runTigger()`. This function takes the two inputs discussed above and can perform any combination of the following:

1. Infer the presence of novel IGHV alleles not in the germline database
2. Infer the individual's IGHV genotype
3. Correct the IGHV allele calls of the samples based on the IGHV genotype

These options are controlled by the logical arguments `find_novel`, `find_genotype`, and `correct_calls`, which default to `TRUE`. Note, however, that if `find_genotype` is `FALSE`, the provided germline database should contain only the alleles carried by the individual from which the Rep-Seq data was isolated. See the help file (`?runTigger`) for a description of all options.

```
# Run tigger on the example input
sample_output = runTigger(sample_db, germline_ighv)
```

```
## Finding novel alleles...done.
## Finding genotype...done.
## Correcting allele calls...done.
```

`sample_output` should now be a list of length three with the names `novel`, `genotype`, and `new_calls`. Let's take a moment to examine the contents of each. (If any of these three options of `runTigger` are set to `FALSE`, the some elements of `sample_output` will be empty.)

Novel Alleles

Potential novel alleles detected by TIgGER are in the first element of the list. Some of these may have been included in the genotype (see below) while others may not have been. A summary of which were included will be printed by `novelSummary()`, which will also return the named novel nucleotide sequences. You may wish to add these to the existing germline database alleles. However, you may also set `seqs_to_return` to `"all"` to also return the non-genotype potential novel alleles.

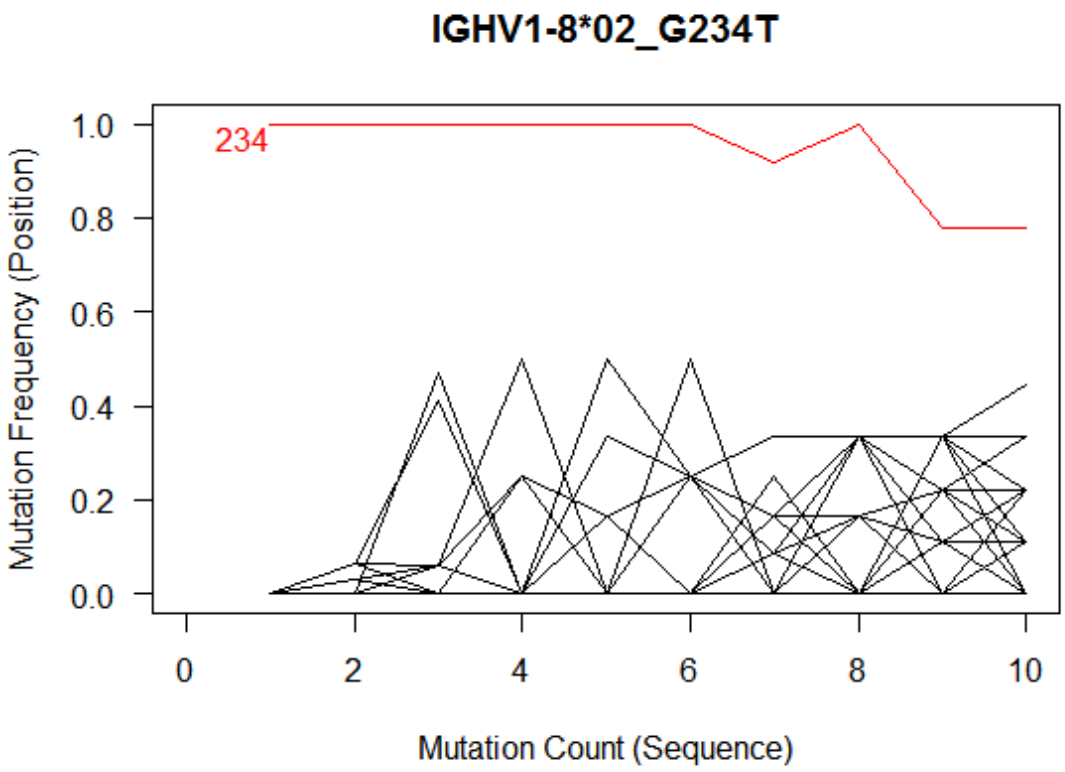
```
# Summarize the detected novel alleles
novel_sequences = novelSummary(sample_output, seqs_to_return = "in genotype")

## 1 potential novel alleles were detected.
## 1 novel alleles were common enough to be included in the genotype:
##  IGHV1-8*02_G234T

germline_ighv = c(germline_ighv, novel_sequences)
```

The first part of IGHV1-8*02_G234T identifies the nearest database allele, and the part after the underscore indicates the polymorphic position(s) using the format `[database_nucleotide][position][polymorphic_nucleotide]`.

The TIgGER procedure for identifying novel alleles (see citation above) involves taking all seqeunces which align to a particular germline allele and, for each position along the aligned sequences, plotting the mutation frequency at that postion as a function of the sequence-wide mutation count. While mutational hot-spots and cold-spots are both expected to have a *y*-intercept around zero, polymorphic positions will have a *y*-intercept larger than zero. The required minimum *y*-intercept may be specified in `runTigger()` by `y_intercept`, but defaults to `1/8`.

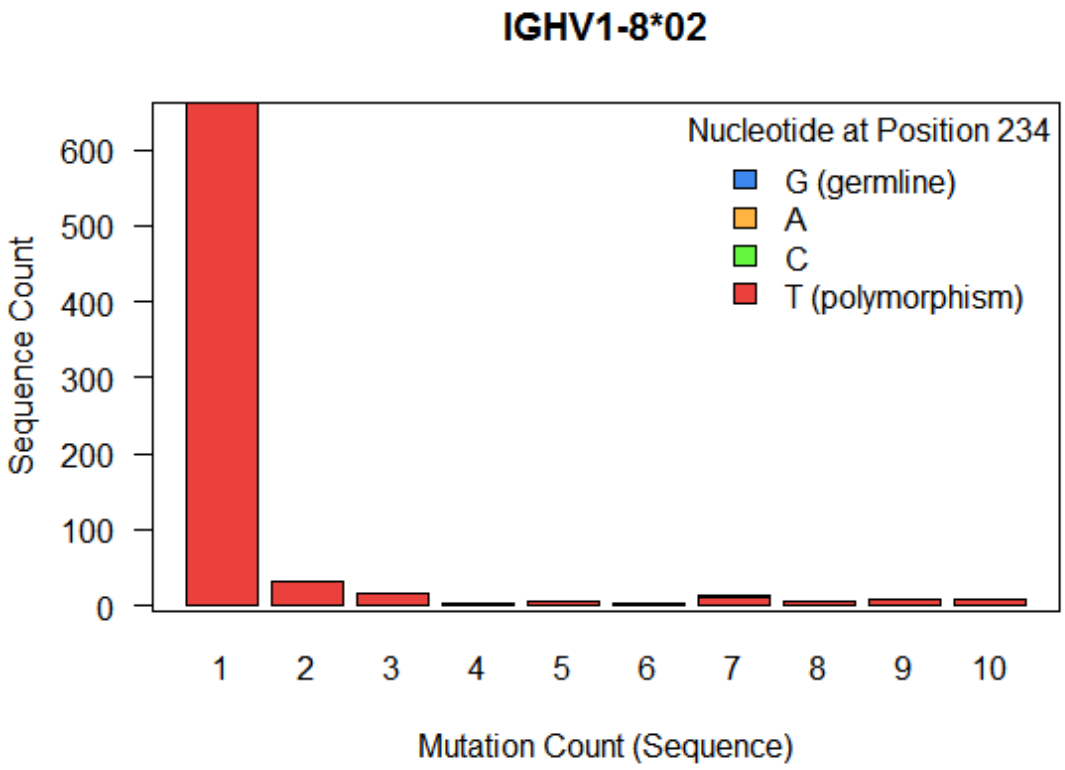
Passing this *y*-intercept threshold is the first of three pieces of evidence that support the novel allele. To view the plots of positional mutation frequency vs sequence-wide mutation count, `plotNovelLines()` may be used. Polymorphic positions are highlighted in red. Remember that some or all sequences may not have been included in the final genotype.

```
# Plot positional mutation frequency versus sequence-wide mutation count
plotNovelLines(sample_output$novel)
```



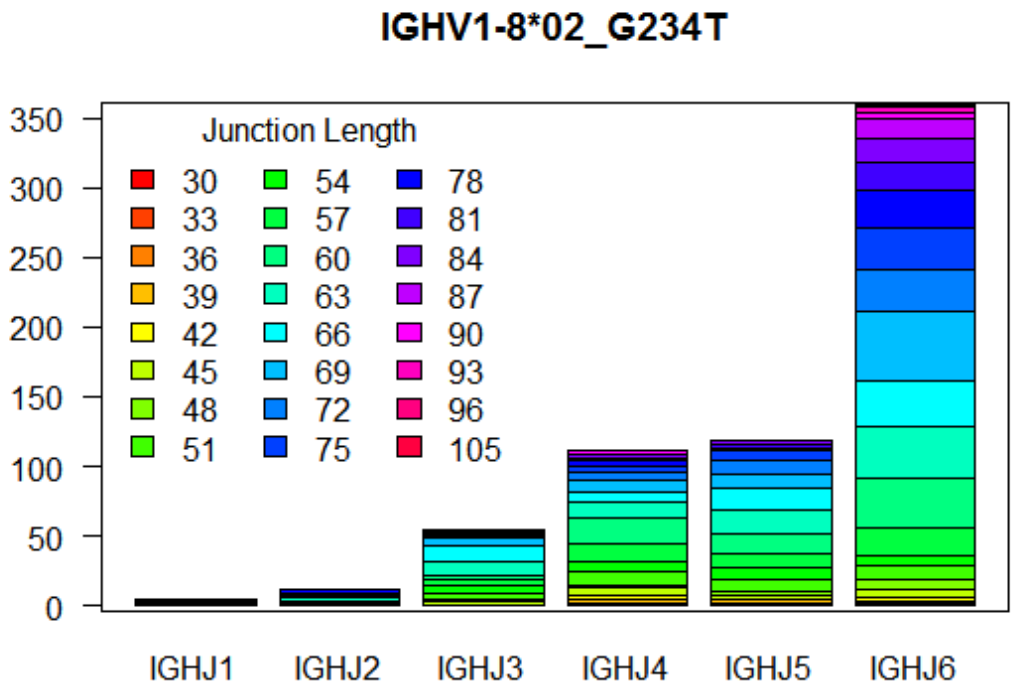
The second piece of evidence supporting novel allele calls is the nucleotide usage at the polymorphic positions as a function of sequence-wide mutation count. We expect the polymorphic allele to be prevalent at all mutation counts, and we expect the mutation count equal to the number of polymorphisms in the novel sequence to be the most prevalent. `plotNovelBars()` can be used to view the nucleotide usage at polymorphic positions.

```
# Plot nucleotide usage at polymorphic positions
plotNovelBars(sample_output$novel)
```



Finally, to avoid cases where a clonal expansion might lead to a false positive, combinations of J gene and junction length are examined among sequences which *perfectly match* the proposed germline allele. A true novel allele is expected to utilize a wide range of J genes, and sequences with different junction length can be ruled out as not being clonally related. The maximum portion of sequences which can consist of a specific combination of J gene and junction length may be specified in `runTigger()` by `j_max`. `plotJunctionBars()` can be used to view the distribution of J genes and junctions.

```
# Plot J and junction usage for sequences perfectly matching the novel allele
plotJunctionBars(sample_output$novel)
```



Genotype

The second element of the list output by `runTigger()` is the genotype. This is the collection of alleles inferred to be carried by the subject. For each gene, `runTigger()` finds the sequences which might be interpreted as unmutated (as these should have the least error in their allele assignment) and then identifies the smallest set of alleles that could explain the majority of these calls. (This "majority" is set ot `7/8` but may be controlled by modifying `fraction_to_explain` in the function `runTigger()`. Additionally, genes representing fewer than `0.1%` of sequences are, by default, excluded; this can be controlled by the user using `gene_cutoff`.)

The genotype output is designed to be human readable. For each allele, the number of sequences which perfectly match the germline are listed int he same order as the alleles are listed. The total number of sequences that perfectly match any allele of that gene is also given. Note that novel alleles are indicated using the same nomenclature as above.

```
# View the inferred genotype
print(sample_output$genotype)
```

```
##         gene     alleles         counts total
## 1    IGHV1-2       02,04      1542,669  2211
## 2    IGHV1-3          01           532   532
## 3    IGHV1-8 01,02_G234T      1015,661  1676
## 4   IGHV1-18          01          2113  2113
```

```
## 5   IGHV1-24          01             493   493
## 6   IGHV1-46          01            1220  1220
## 7   IGHV1-58       01,02           53,49   102
## 8   IGHV1-69    01,06,04 1994,1179,1027  4223
## 9 IGHV1-69-2          01             101   101
```

`genotypeFasta()` is useful is you wish to ouput the sequences rather than the allele names. Given a genotype and the vector of database germline sequences, it will return a vector of germline alleles carried by the individual. Users may use `writeFasta()` to save the sequences to a fasta file.

```
# Get the nucleotide sequences of all genotype alleles
genotype_sequences = genotypeFasta(sample_output$genotype, germline_ighv)
```

Corrected Allele Calls

Finally, the original V allele calls have been limited to only those within the inferred genotype; the third element of the output list contains a vector of allele calls equal in length and ordering to the original allele calls. This can be combined to the existing data frame, and used in future analyses as follows.

```
# Extract the corrected V allele calls and appened them to the data frame
V_CALL_GENOTYPED = sample_output$new_calls
sample_db = cbind(sample_db, V_CALL_GENOTYPED)
```

References

1. Alamyar *et al.* (2010)
2. Munshaw and Kepler (2010)
3. Lefranc *et al.* (2003)