Analysis of high-throughput sequencing of T and B cell receptors with LymphoSeq

David G. Coffey, MD

2016-10-17

Application of high-throughput sequencing of T and B lymphocyte antigen receptors has great potential for improving the monitoring of lymphoid malignancies, assessing immune reconstitution after hematopoietic stem cell transplantation, and characterizing the composition of lymphocyte repertoires.[1] LymhoSeq is an R package designed to import, analyze, and visualize antigen receptor sequencing from Adaptive Biotechnologies' ImmunoSEQ assay.[2] The package is also adaptable to the analysis of T and B cell receptor sequencing processed using other platforms such as MiXCR[3] or IMGT/HighV-QUEST.[4] This vignette has been written to highlight some of the features of LymphoSeq and guide the user through a typical workflow.

Importing data

The LymphoSeq function `readImmunoSeq` imports tab-separated value (.tsv) files exported by Adaptive Biotechnologies ImmunoSEQ analyzer where each row represents a unique sequence and each column is a variable with information about that sequence such as read count, frequency, or variable gene name. Only files with the extension `.tsv` are imported while all other are disregarded. It is possible to import files processed using other platforms as long as the files are tab-delimited with the extension .tsv and have identical column names as the ImmunoSEQs files (see `readImmunoSeq` manual for a list of column names).[5]

In the example below, `system.file("extdata", "TCRB_sequencing", package = "LymphoSeq")` refers to a path to a directory embedded within the LymphoSeq package containing an example data set of T cell receptor beta (TCRB) sequencing from the peripheral blood of a patient who underwent a bone marrow transplant. The files are named according to the day posttransplant and the immunophenotype of the T cells if sorted prior to sequencing. The function `readImmunoSeq` is used to import the files in as a list object where each file becomes a data frame. You can import all columns from each file by setting the `columns` parameter to `"all"` or list just those columns you are interested in as shown below. Be aware that Adaptive Biotechnologies has changed the column names of their files over time and if the headings of your files are not all the same, you will need to specify `"all"` or provide all variations of the column header as done for "count" and "frequencyCount" in the example below. If the directory you are pointing to contains subdirectories with additional sequencing files, you can choose to import those by setting the parameter `recursive` to `TRUE`, otherwise only the files in the top directory will be imported.

```
library(LymphoSeq)
```

```
## Loading required package: LymphoSeqDB
```

```
file.path <- system.file("extdata", "TCRB_sequencing", package = "LymphoSeq")

file.list <- readImmunoSeq(path = file.path,
                           columns = c("aminoAcid", "nucleotide", "count", "count (templates)",
                                       "count (reads)", "frequencyCount", "frequencyCount (%)",
                                       "estimatedNumberGenomes", "vFamilyName", "dFamilyName",
                                       "jFamilyName"),
                           recursive = FALSE)
```

Notice that each data frame listed in the `file.list` object is named according the ImmunoSEQ file name files. If different names are desired, you may rename the original `.tsv` files or assign `names(file.list)` to a new character vector of desitred names within R.

[1] Warren, E. H. *et al. Blood* 2013;122:19–22.
[2] http://www.adaptivebiotech.com/immunoseq
[3] http://mixcr.milaboratory.com
[4] http://www.imgt.org/HighV-QUEST
[5] Some functionality may be lost. Extensive testing with other platforms has not been performed.

```r
names(file.list)
```

```
 [1] "TCRB_Day0_Unsorted"    "TCRB_Day1320_CD8_CMV"
 [3] "TCRB_Day1320_Unsorted" "TCRB_Day32_Unsorted"
 [5] "TCRB_Day369_CD8_CMV"   "TCRB_Day369_Unsorted"
 [7] "TCRB_Day83_CD8_CMV"    "TCRB_Day83_Unsorted"
 [9] "TCRB_Day949_CD4"       "TCRB_Day949_CD8"
[11] "TCRB_Day949_Unsorted"
```

Having the data in the form of a list makes it easy to apply a function over that list using the base functions `sapply` (outputs a matrix) or `lapply` (outputs a list). For example, you may use the function `dim` to report the dimensions of each data frame as shown below. Notice that in the example data set, each data frame has less than 1000 rows (e.g. sequences) since it has been truncated from its original size for demonstration purposes. There are 8 columns to each data frame as specified by the `readImmunoSeq` function. In place of `dim`, you may also use `colnames`, `nrow`, `ncol`, or other more complex functions that perform operations on subsetted columns.

```r
sapply(file.list, dim)
```

```
     TCRB_Day0_Unsorted TCRB_Day1320_CD8_CMV TCRB_Day1320_Unsorted
[1,]                999                   40                   999
[2,]                  8                    8                     8
     TCRB_Day32_Unsorted TCRB_Day369_CD8_CMV TCRB_Day369_Unsorted
[1,]                 920                 414                  999
[2,]                   8                   8                    8
     TCRB_Day83_CD8_CMV TCRB_Day83_Unsorted TCRB_Day949_CD4
[1,]                201                 999             999
[2,]                  8                   8               8
     TCRB_Day949_CD8 TCRB_Day949_Unsorted
[1,]             999                  999
[2,]               8                    8
```

Subsetting data

If you imported all of the files from your project but just want to perform an analysis on a subset, use standard R methods to subset the list. Remember that a single bracket `[` returns a list and a double bracket `[[` returns a single data frame.

```r
CMV <- file.list[grep("CMV", names(file.list))]
names(CMV)
```

```
[1] "TCRB_Day1320_CD8_CMV" "TCRB_Day369_CD8_CMV"  "TCRB_Day83_CD8_CMV"
```

```r
TCRB_Day0_Unsorted <- file.list[["TCRB_Day0_Unsorted"]]
head(TCRB_Day0_Unsorted)
```

```
                                                                       nucleotide
1 TCAATTCCCTGGAGCTTGGTGACTCTGCTGTGTATTTCTGTGCCAGCAGCCATCGGGACAGAGAACACTGAAGCTTTCTTTGGACAA
2 CTGATTCTGGAGTCCGCCAGCACCAACCAGACATCTATGTACCTCTGTGCCAGCAGCCCCGTGAGCAATGAGCAGTTCTTCGGGCCA
3 ATCAATTCCCTGGAGCTTGGTGACTCTGCTGTGTATTTCTGTGCCAGCAGCCAAGAAGTTCCGCCTTACCAAGCTTTCTTTGGACAA
4 TGCCATCCCCAACCAGACAGCTCTTTACTTCTGTGCCACCAGTGTCCACAAACAGGGGGCAGGACCGGGGAGCTGTTTTTTGGAGAA
5 CACACCCTGCAGCCAGAAGACTCGGCCCTGTATCTCTGCGCCAGCAGCCAAGAGGCTAGCGGGAGACAGACCCAGTACTTCGGGCCA
6 GCCAGCACCAACCAGACATCTATGTACCTCTGTGCCAGCAGTTTGGAGCACACGGGTGCAACTAATGAAAAACTGTTTTTTGGCAGT
        aminoAcid count frequencyCount vFamilyName dFamilyName
1                 89285       6.606637     TCRBV03     TCRBD01
2     CASSPVSNEQFF 50511       3.737558     TCRBV28     TCRBD02
3   CASSQEVPPYQAFF 49129       3.635297     TCRBV03
4                 43293       3.203462     TCRBV24     TCRBD01
```

```
5   CASSQEASGRQTQYF 43264       3.201317     TCRBV04     TCRBD02
6 CASSLEHTGATNEKLFF 40119       2.968602     TCRBV28     TCRBD02
  jFamilyName estimatedNumberGenomes
1     TCRBJ01                   1450
2     TCRBJ02                    822
3     TCRBJ01                    797
4     TCRBJ02                    702
5     TCRBJ02                    704
6     TCRBJ01                    653
```

For more complex subsetting, you can use a metadata file where one column contains the file names and the other columns have additional information about the sample files (e.g pretreatment and posttreatment). You can then subset the metadata file using criteria from the other columns to give you just a character vector of file names that you can use to subset file.list.

```
metadata <- read.csv(system.file("extdata", "metadata.csv", package = "LymphoSeq"))
metadata
```

```
                 samples  day     timePoint phenotype
1     TCRB_Day0_Unsorted    0 Pretransplant  Unsorted
2    TCRB_Day32_Unsorted   32       1 Month  Unsorted
3    TCRB_Day83_Unsorted   83      3 Months  Unsorted
4     TCRB_Day83_CD8_CMV   83      3 Months  CD8+CMV+
5   TCRB_Day369_Unsorted  369        1 Year  Unsorted
6    TCRB_Day369_CD8_CMV  369        1 Year  CD8+CMV+
7   TCRB_Day949_Unsorted  949       2 Years  Unsorted
8        TCRB_Day949_CD4  949       2 Years      CD4+
9        TCRB_Day949_CD8  949       2 Years      CD8+
10 TCRB_Day1320_Unsorted 1320       3 Years  Unsorted
11  TCRB_Day1320_CD8_CMV 1320       3 Years  CD8+CMV+
```

```
selected <- as.character(metadata[metadata$phenotype == "Unsorted" &
                                  metadata$day > 300, "samples"])
file.list.selected <- file.list[selected]
names(file.list.selected)
```

```
[1] "TCRB_Day369_Unsorted"  "TCRB_Day949_Unsorted"  "TCRB_Day1320_Unsorted"
```

Extracting productive sequences

A productive sequence is defined as a sequences that is in frame and does not have an early stop codon. If you sequenced genomic DNA as opposed to complimentary DNA made from RNA, then you will have unproductive and productive sequences in your data files. Use the function `productiveSeq` to remove unproductive sequences and recompute the frequencyCount for each of your samples.

If you are interested in just the complementarity determining region 3 (CDR3) amino acid sequences, then set aggregate to `"aminoAcid"` and the count and estimated number of genomes for duplicate amino acid sequences will be summed. Note that the resulting list of data frames will have columns corresponding to "aminoAcid", "count", "frequencyCount", and "estimatedNumberGenomes" (if this column is available) only. All other columns, such as those corresponding to the V, D, and J gene names, will be removed if they were included in your original file list. The reason for this is to avoid confusion since a single amino acid CDR3 sequence may be encoded by multiple different nucleotide sequences with differing V, D, and J genes.

```
productive.aa <- productiveSeq(file.list = file.list, aggregate = "aminoAcid",
                               prevalence = FALSE)
```

Alternatively, you may set aggregate to `"nucleotide"` and the resulting list of data frames will all have the same columns as your original file list. Take note that some LymphoSeq functions require a productive sequence list aggregated by amino acid or nucleotide.

```
productive.nt <- productiveSeq(file.list = file.list, aggregate = "nucleotide",
                               prevalence = FALSE)
```

If the parameter `prevalence` is set to `TRUE`, then a new column is added to each of the data frames giving the prevalence (%) of each CDR3 amino acid sequence in 55 healthy donor peripheral blood samples. Values range from 0 to 100% where 100% means the sequence appeared in the blood of all 55 individuals. The data for this operation resides in a separate package that is automatically loaded called LymphoSeqDB. Please refer to that package manual for more details.

Notice in the example below that there are no amino acid sequences given in the first and fourth row of the file.list data frame for sample "TCRB_Day949_Unsorted". This is because the nucleotide sequence is out of frame and does not produce a productively transcribed amino acid sequence. If an asterisk (*) appears in the amino acid sequences, this would indicate an early stop codon.

```
head(file.list[["TCRB_Day0_Unsorted"]])
```

```
                                                                                                nucleotide
1 TCAATTCCCTGGAGCTTGGTGACTCTGCTGTGTATTTCTGTGCCAGCAGCCATCGGGACAGAGAACACTGAAGCTTTCTTTGGACAA
2 CTGATTCTGGAGTCCGCCAGCACCAACCAGACATCTATGTACCTCTGTGCCAGCAGCCCCGTGAGCAATGAGCAGTTCTTCGGGCCA
3 ATCAATTCCCTGGAGCTTGGTGACTCTGCTGTGTATTTCTGTGCCAGCAGCCAAGAAGTTCCGCCTTACCAAGCTTTCTTTGGACAA
4 TGCCATCCCCAACCAGACAGCTCTTTACTTCTGTGCCACCAGTGTCCACAAACAGGGGGCAGGACCGGGGAGCTGTTTTTTGGAGAA
5 CACACCCTGCAGCCAGAAGACTCGGCCCTGTATCTCTGCGCCAGCAGCCAAGAGGCTAGCGGGAGACAGACCCAGTACTTCGGGCCA
6 GCCAGCACCAACCAGACATCTATGTACCTCTGTGCCAGCAGTTTGGAGCACACGGGTGCAACTAATGAAAAACTGTTTTTTGGCAGT
          aminoAcid count frequencyCount vFamilyName dFamilyName
1                   89285       6.606637     TCRBV03     TCRBD01
2      CASSPVSNEQFF 50511       3.737558     TCRBV28     TCRBD02
3    CASSQEVPPYQAFF 49129       3.635297     TCRBV03
4                   43293       3.203462     TCRBV24     TCRBD01
5   CASSQEASGRQTQYF 43264       3.201317     TCRBV04     TCRBD02
6 CASSLEHTGATNEKLFF 40119       2.968602     TCRBV28     TCRBD02
  jFamilyName estimatedNumberGenomes
1     TCRBJ01                   1450
2     TCRBJ02                    822
3     TCRBJ01                    797
4     TCRBJ02                    702
5     TCRBJ02                    704
6     TCRBJ01                    653
```

After `productiveSeq` is run, the unproductive sequences are removed and the frequencyCount is recalculated for each sequence. If there were two identical amino acid sequences that differed in their nucleotide sequence, they would be combined and their counts added together.

```
head(productive.aa[["TCRB_Day0_Unsorted"]])
```

```
          aminoAcid count frequencyCount estimatedNumberGenomes
1      CASSPVSNEQFF 50511       5.543769                    822
2    CASSQEVPPYQAFF 49129       5.392090                    797
3   CASSQEASGRQTQYF 43264       4.748384                    704
4 CASSLEHTGATNEKLFF 40119       4.403209                    653
5       CASSPGDEQYF 38100       4.181616                    619
6  CSARSPSTGTLAEAFF 26377       2.894973                    429
```

Finally, notice that the productive.nt data frame for sample "TCRB_Day949_Unsorted" below has additional columns not present in productive.aa but are in file.list. This is because the data frame was aggregated by nucleotide sequence and all of the original columns from file.list were carried over.

```
head(productive.nt[["TCRB_Day0_Unsorted"]])
```

```
                                                                                 nucleotide
1 CTGATTCTGGAGTCCGCCAGCACCAACCAGACATCTATGTACCTCTGTGCCAGCAGCCCCGTGAGCAATGAGCAGTTCTTCGGGCCA
2 ATCAATTCCCTGGAGCTTGGTGACTCTGCTGTGTATTTCTGTGCCAGCAGCCAAGAAGTTCCGCCTTACCAAGCTTTCTTTGGACAA
3 CACACCCTGCAGCCAGAAGACTCGGCCCTGTATCTCTGCGCCAGCAGCCAAGAGGCTAGCGGGAGACAGACCCAGTACTTCGGGCCA
4 GCCAGCACCAACCAGACATCTATGTACCTCTGTGCCAGCAGTTTGGAGCACACGGGTGCAACTAATGAAAAACTGTTTTTTGGCAGT
5 CCCCTGACCCTGGAGTCTGCCAGGCCCTCACATACCTCTCAGTACCTCTGTGCCAGCAGTCCGGGGGACGAGCAGTACTTCGGGCCG
6 AGTGCCCATCCTGAAGACAGCAGCTTCTACATCTGCAGTGCTAGATCACCCAGTACAGGGACCCTCGCTGAAGCTTTCTTTGGACAA
          aminoAcid count frequencyCount vFamilyName dFamilyName
1      CASSPVSNEQFF 50511       5.543769     TCRBV28     TCRBD02
2    CASSQEVPPYQAFF 49129       5.392090     TCRBV03
3   CASSQEASGRQTQYF 43264       4.748384     TCRBV04     TCRBD02
4 CASSLEHTGATNEKLFF 40119       4.403209     TCRBV28     TCRBD02
5       CASSPGDEQYF 38100       4.181616     TCRBV25     TCRBD02
6  CSARSPSTGTLAEAFF 26377       2.894973     TCRBV20     TCRBD01
  jFamilyName estimatedNumberGenomes
1     TCRBJ02                    822
2     TCRBJ01                    797
3     TCRBJ02                    704
4     TCRBJ01                    653
5     TCRBJ02                    619
6     TCRBJ01                    429
```

Create a table of summary statistics

To create a table summarizing the total number of sequences, number of unique productive sequences, number of genomes, entropy, clonality, Gini coefficient, and the frequency (%) of the top productive sequence in each imported file, use the function `clonality`.

```
clonality(file.list = file.list)
```

```
                 samples totalSequences uniqueProductiveSequences
1        TCRB_Day949_CD4            999                       845
2        TCRB_Day949_CD8            999                       796
3     TCRB_Day0_Unsorted            999                       837
4     TCRB_Day83_CD8_CMV            201                       122
5    TCRB_Day32_Unsorted            920                       767
6    TCRB_Day369_CD8_CMV            414                       281
7    TCRB_Day83_Unsorted            999                       830
8   TCRB_Day1320_CD8_CMV             40                        25
9   TCRB_Day369_Unsorted            999                       828
10  TCRB_Day949_Unsorted            999                       831
11 TCRB_Day1320_Unsorted            999                       833
   totalGenomes totalCount  entropy  clonality giniCoefficient
1         25767    1795561 5.597259 0.42431656       0.8486265
2         26236    2161314 5.535828 0.42554289       0.9018374
3         18215    1158510 7.144318 0.26416150       0.7956718
4           254       4553 5.891883 0.14989093       0.6133629
5            NA      31078 8.296630 0.13424209       0.6007820
6          1794      52480 5.011083 0.38396606       0.8677666
7            NA     427427 7.285945 0.24863671       0.7100983
8            53         53 4.486348 0.03391748       0.2313514
9            NA     725668 6.037979 0.37710971       0.8043414
10         6547    1486480 6.774799 0.30147383       0.7739322
11       180079     180079 5.708193 0.41165830       0.8845652
   topProductiveSequence
1              29.232143
2              19.131268
3               5.543769
4               8.917656
```

```
5               4.865016
6              17.718550
7              13.821673
8              10.810811
9              17.568430
10             13.510800
11             14.422142
```

The clonality score is derived from the Shannon entropy, which is calculated from the frequencies of all productive sequences divided by the logarithm of the total number of unique productive sequences. This normalized entropy value is then inverted (1 - normalized entropy) to produce the clonality metric.

The Gini coefficient[6] is an alternative metric used to calculate repertoire diversity and is derived from the Lorenz curve. The Lorenz curve is drawn such that x-axis represents the cumulative percentage of unique sequences and the y-axis represents the cumulative percentage of reads.[7] A line passing through the origin with a slope of 1 reflects equal frequencies of all clones. The Gini coefficient is the ratio of the area between the line of equality and the observed Lorenz curve over the total area under the line of equality.

Both Gini coefficient and clonality are reported on a scale from 0 to 1 where 0 indicates all sequences have the same frequency and 1 indicates the repertoire is dominated by a single sequence.

Searching for sequences

To search for one or more amino acid or nucleotide CDR3 sequences in a list of data frames, use the function `searchSeq`. You may specify to search in either a list of productive or unproductive data frames.

```
searchSeq(list = productive.aa, sequence = "CASSPVSNEQFF", type = "aminoAcid",
          match = "global", editDistance = 0)
```

```
                  sample    aminoAcid count frequencyCount
1     TCRB_Day0_Unsorted CASSPVSNEQFF 50511     5.54376923
260 TCRB_Day369_Unsorted CASSPVSNEQFF   211     0.03315421
64    TCRB_Day83_CD8_CMV CASSPVSNEQFF    10     0.35958288
334      TCRB_Day949_CD8 CASSPVSNEQFF   214     0.01159812
    estimatedNumberGenomes
1                      822
260                      0
64                       1
334                      2
```

If you have only a partial sequence, set the parameter match to `"partial"`. If you are looking for related sequences that differ by one or more nucleotides or amino acids, then increase the `editDistance` value. Edit distance is a way of quantifying how dissimilar two sequences are to one another by counting the minimum number of operations required to transform one sequence into the other. For example, an edit distance of 0 means the sequences are identical and an edit distance of 1 indicates that the sequences differ by a single amino acid or nucleotide.

Searching for published T cell sequences with known antigen specificity

To search your entire list of data frames for a published amino acid CDR3 TCRB sequence with known antigen specificity, use the function `searchPublished`.

```
published <- searchPublished(list = productive.aa)
head(published)
```

[6] https://en.wikipedia.org/wiki/Gini_coefficient
[7] https://en.wikipedia.org/wiki/Lorenz_curve

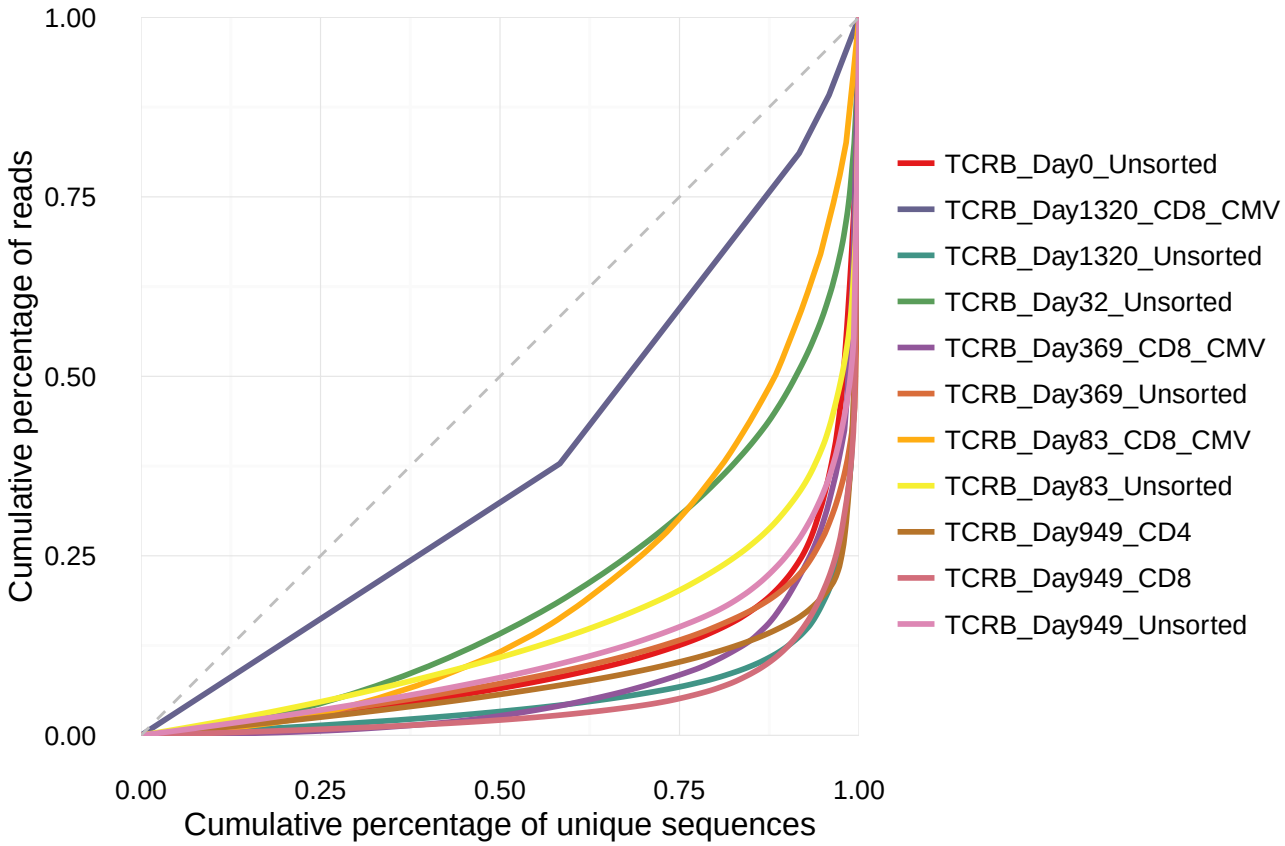
```
                 sample     aminoAcid count frequencyCount
1   TCRB_Day32_Unsorted CASASSGTDTQYF    33    0.131594688
2  TCRB_Day949_Unsorted  CASSFSTDTQYF   279    0.023538126
3 TCRB_Day1320_Unsorted CASSIRSAYEQYF    15    0.010202902
4       TCRB_Day949_CD8 CASSIRSAYEQYF   593    0.032138727
5   TCRB_Day32_Unsorted CASSLAPSYEQYF    17    0.067791203
6 TCRB_Day1320_Unsorted  CASSLGEQPQHF    10    0.006801934
  estimatedNumberGenomes     PMID         HLA  antigen         epitope
1                      0 20647322 HLA-A*24:02 Leukemia            <NA>
2                      2 23267020    HLA-A*02      EBV BMFL1-GLCTLVAML
3                     15 21048112    HLA-A*02      EBV BMLF1-GLCTLVAML
4                      7 21048112    HLA-A*02      EBV BMLF1-GLCTLVAML
5                      0 23267020    HLA-B*08      EBV  BZLF1-RAKFKQLL
6                     10 23521884 HLA-B*27:05      HIV KK10-KRWIILGLNK
  prevalence
1        7.3
2       85.5
3       18.2
4       18.2
5       69.1
6       29.1
```

For each found sequence, a table is provides listing the antigen, epitope, HLA type, PubMed ID (PMID), and prevalence (%) of the sequence among 55 healthy donor blood samples. The data for this function resides in the separate LymphoSeqDB package that is automatically loaded when the function is called. Please refer to that package manual for more details.

Visualizing repertoire diversity

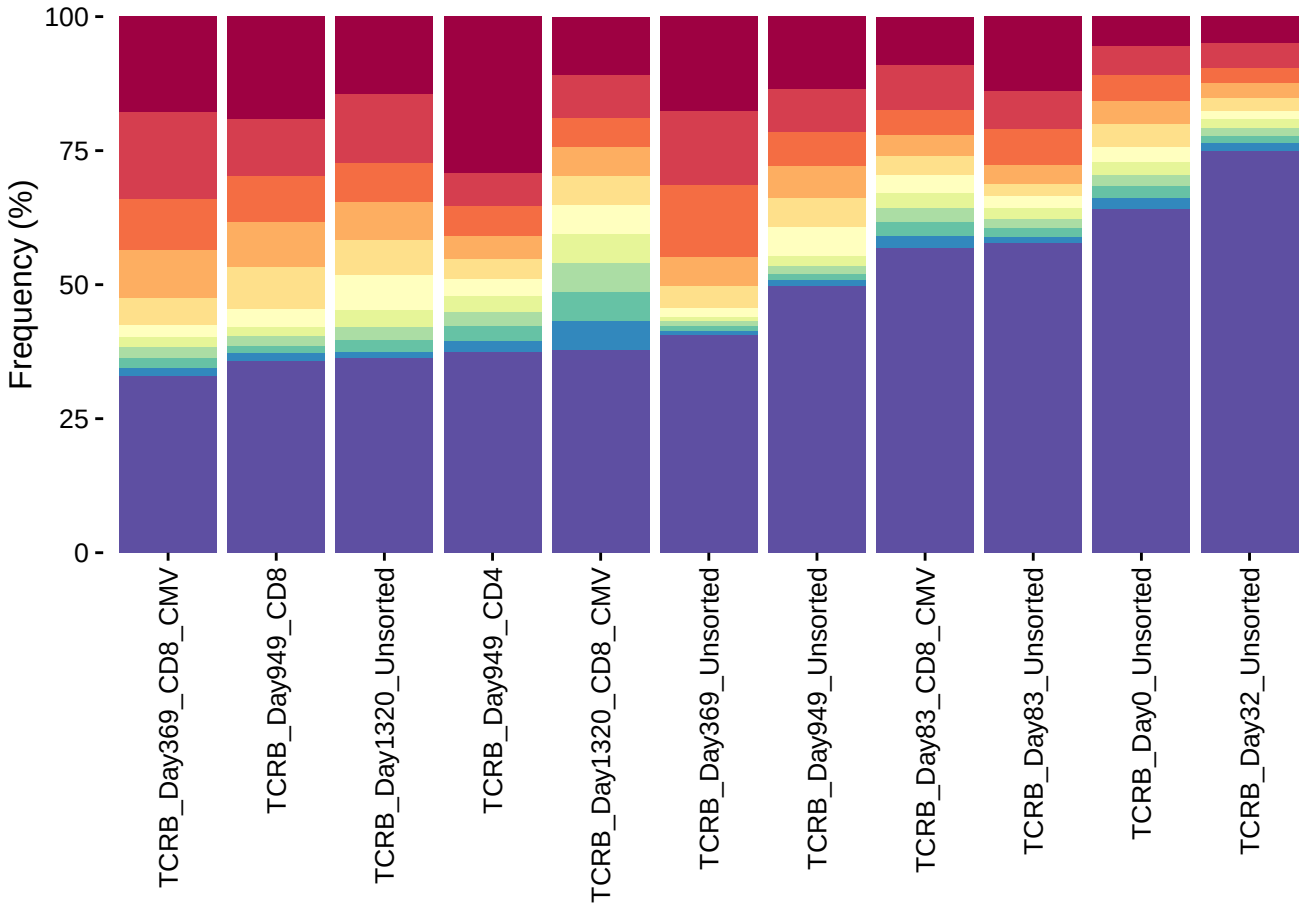
Antigen receptor repertoire diversity can be characterized by a number such as clonality or Gini coefficient calculated by the `clonality` function. Alternatively, you can visualize the repertoire diversity by plotting the Lorenz curve for each sample as defined above. In this plot, the more diverse samples will appear near the dotted diagonal line (the line of equality) whereas the more clonal samples will appear to have a more bowed shape.

```
lorenzCurve(samples = names(productive.aa), list = productive.aa)
```



Alternatively, you can get a feel for the repertoire diversity by plotting the cumulative frequency of a selected number of the top most frequent clones using the function `topSeqsPlot`. In this case, each of the top sequences are represented by a different color and all less frequent clones will be assigned a single color (violet).

```
topSeqsPlot(list = productive.aa, top = 10)
```



Both of these functions are built using the ggplot2 package[8]. You can reformat the plot using ggplot2 functions. Please refer to the `lorenzCurve` and `topSeqsPlot` manual for specific examples.

Comparing samples

To compare the T or B cell repertoires of all samples in a pairwise fashion, use the `bhattacharyyaMatrix` or `similarityMatrix` functions. Both the Bhattacharyya coefficient and similarity score are measures of the amount of overlap between two samples. The value for each ranges from 0 to 1 where 1 indicates the sequence frequencies are identical in the two samples and 0 indicates no shared frequencies. The Bhattacharyya coefficient differs from the similarity score in that it involves weighting each shared sequence in the two distributions by the arithmetic mean of the frequency of each sequence, while calculating the similarity scores involves weighting each shared sequence in the two distributions by the geometric mean of the frequency of each sequence in the two distributions.

```
bhattacharyya.matrix <- bhattacharyyaMatrix(productive.seqs = productive.aa)
bhattacharyya.matrix[,1:2]
```

```
                    TCRB_Day1320_Unsorted TCRB_Day949_Unsorted
TCRB_Day1320_Unsorted            1.00000000           0.88013589
TCRB_Day949_Unsorted             0.88013589           1.00000000
TCRB_Day369_Unsorted             0.84210238           0.77225070
TCRB_Day1320_CD8_CMV             0.46204244           0.39421126
TCRB_Day83_Unsorted              0.62582845           0.58334619
TCRB_Day369_CD8_CMV              0.79605192           0.72972153
TCRB_Day32_Unsorted              0.32118959           0.27850843
TCRB_Day83_CD8_CMV               0.42168018           0.40854427
TCRB_Day0_Unsorted               0.01700697           0.01416233
TCRB_Day949_CD8                  0.81836252           0.81470456
TCRB_Day949_CD4                  0.44279429           0.42687836
```

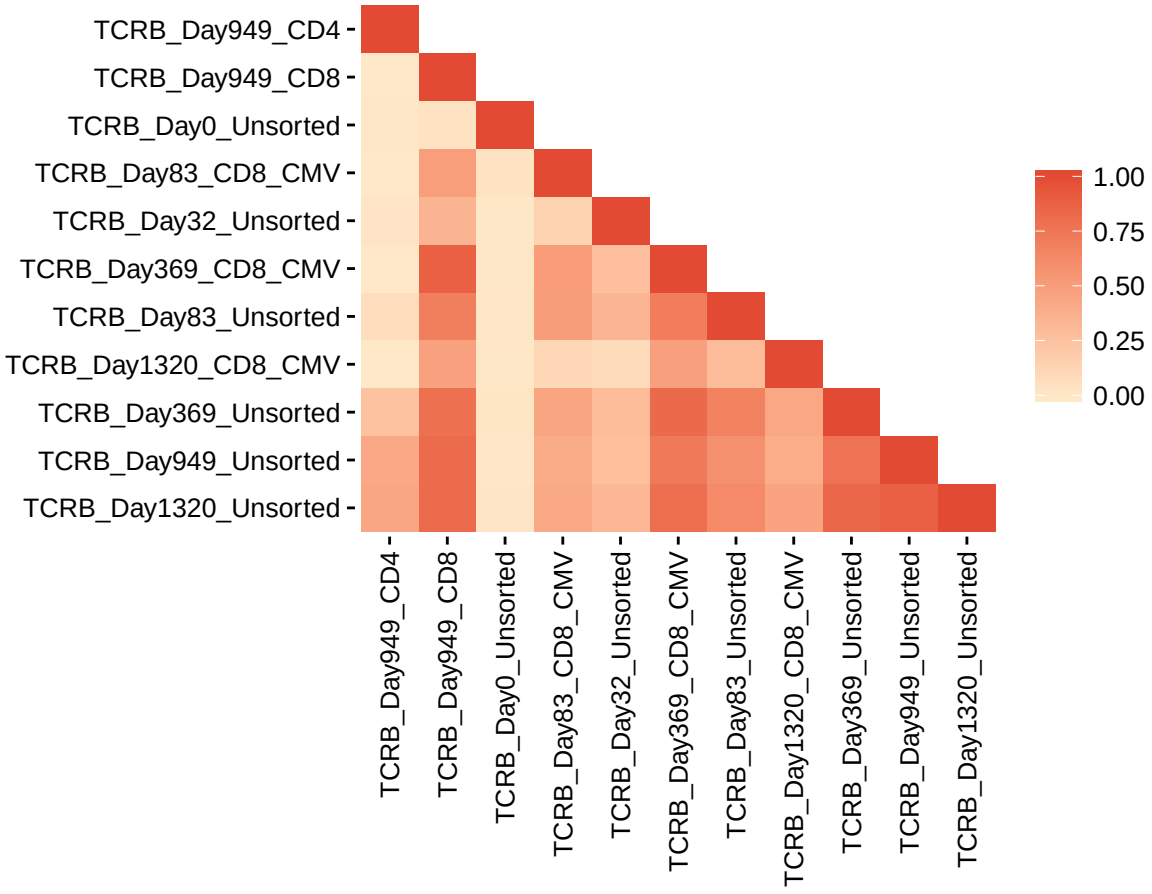
[8] http://www.cookbook-r.com/Graphs/

```
similarity.matrix <- similarityMatrix(productive.seqs = productive.aa)
similarity.matrix[,1:2]
```

```
                    TCRB_Day1320_Unsorted TCRB_Day949_Unsorted
TCRB_Day1320_Unsorted             1.00000000           0.85741424
TCRB_Day949_Unsorted              0.85741424           1.00000000
TCRB_Day369_Unsorted              0.84852643           0.78458455
TCRB_Day1320_CD8_CMV              0.48448189           0.37158455
TCRB_Day83_Unsorted               0.62977574           0.58472269
TCRB_Day369_CD8_CMV               0.75230736           0.59370996
TCRB_Day32_Unsorted               0.40554581           0.37243182
TCRB_Day83_CD8_CMV                0.43445840           0.39263374
TCRB_Day0_Unsorted                0.01231397           0.02166242
TCRB_Day949_CD8                   0.91435304           0.85504038
TCRB_Day949_CD4                   0.80667431           0.54676809
```

The results of either function can be visualized by the `pairwisePlot` function.

```
pairwisePlot(matrix = bhattacharyya.matrix)
```



To view sequences shared between two or more samples, use the function `commonSeqs`. This function requires that a productive amino acid list be specified.

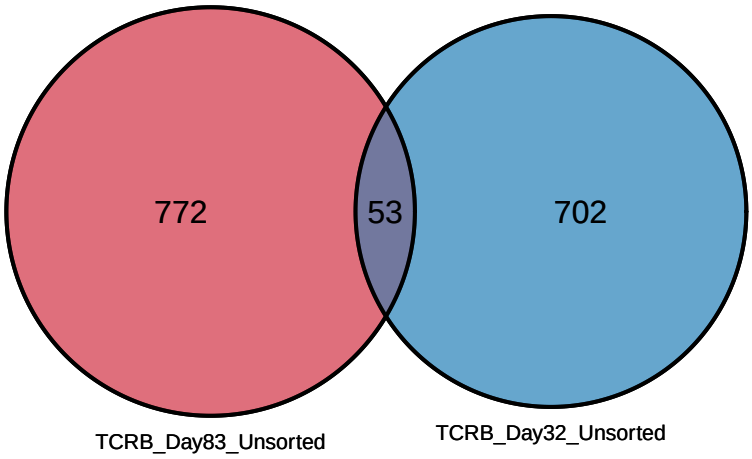
```
common <- commonSeqs(samples = c("TCRB_Day0_Unsorted", "TCRB_Day32_Unsorted"),
                     productive.aa = productive.aa)
head(common)
```

```
        aminoAcid TCRB_Day0_Unsorted TCRB_Day32_Unsorted
1 CASSQDRTGQYGYTF         0.47380673          0.80551900
2    CAWTGGTTEAFF         0.10887567          0.15153328
```

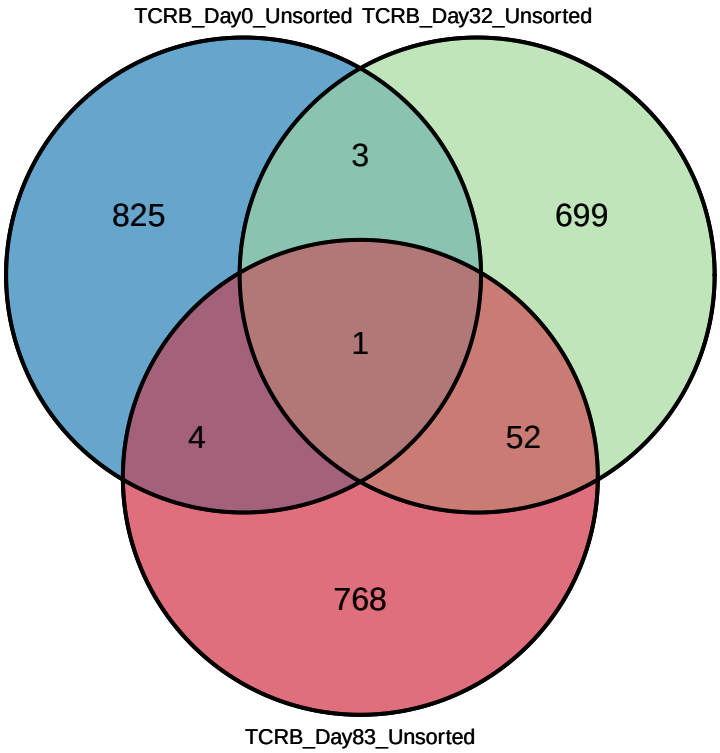
```
3    CAISEGNYGYTF          0.03566995          0.18742274
4   CASSFGIQETQYF          0.01393872          0.09570523
```

To visualize the number of overlapping sequences between two or three samples in the form of a Venn diagram, use the function `commonSeqVenn`.

```
commonSeqsVenn(samples = c("TCRB_Day32_Unsorted", "TCRB_Day83_Unsorted"),
               productive.seqs = productive.aa)
```

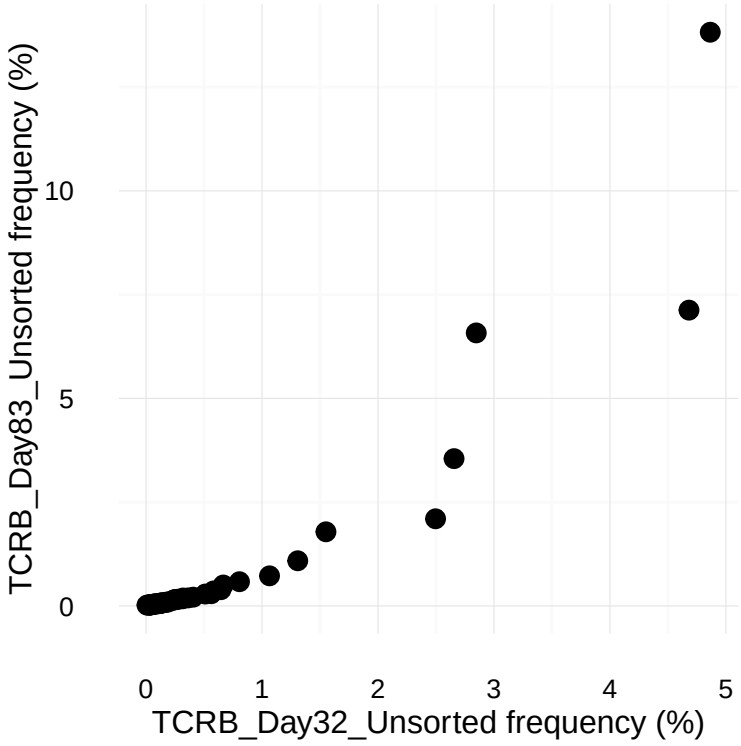



```
commonSeqsVenn(samples = c("TCRB_Day0_Unsorted", "TCRB_Day32_Unsorted", "TCRB_Day83_Unsorted"),
               productive.seqs = productive.aa)
```



To compare the frequency of sequences between two samples as a scatter plot, use the function `commonSeqsPlot`.

```
commonSeqsPlot("TCRB_Day32_Unsorted", "TCRB_Day83_Unsorted",
               productive.aa = productive.aa)
```



Revealing sequences that appear in multiple samples

To create a data frame of unique, productive amino acid sequences as rows and sample names as headers use the `seqMatrix` function. Each value in the data frame represents the frequency that each sequence appears in the sample. You can specify your own list of sequences or all unique sequences in the list using the output of the function `uniqueSeqs`. The `uniqueSeqs` function creates a data frame of all unique, productive sequences and reports the total count in all samples.

```
unique.seqs <- uniqueSeqs(productive.aa = productive.aa)
head(unique.seqs)
```

```
                aminoAcid  count
2465        CASSPAGAYYNEQFF 666188
1070        CASSESAGSTGELFF 430262
2699        CASSPPTGERDTQYF 358432
2998 CASSQDLMTVDSLFAGANVLTF 357006
2154          CASSLQGREKLFF 320305
3103        CASSQDWERLGEQFF 307744
```

```
sequence.matrix <- seqMatrix(productive.aa = productive.aa, sequences = unique.seqs$aminoAcid)
head(sequence.matrix)[1:6]
```

```
             aminoAcid numberSamples TCRB_Day0_Unsorted
1        CASSPAGAYYNEQFF             9          0.0000000
2        CASSPPTGERDTQYF             9          0.0000000
3 CASSQDLMTVDSLFAGANVLTF             9          0.0000000
4        CASSQDRTGQYGYTF             9          0.4738067
5          CASSLQGREKLFF             8          0.0000000
6         CASSQDSSDTEAFF             8          0.0000000
  TCRB_Day1320_CD8_CMV TCRB_Day1320_Unsorted TCRB_Day32_Unsorted
1             8.108108            14.4459484          0.03987718
2             5.405405            12.8039614          0.40674722
3             2.702703             3.0853575          1.55122224
4             0.000000             0.4278417          0.80551900
```

```
5             0.000000             7.2937143          4.86501575
6             0.000000             0.1285566          0.66594888
```

If just the top clones with a frequency greater than a specified amount are of interest to you, then use the `topFreq` function. This creates a data frame of the top productive amino acid sequences having a minimum specified frequency and reports the minimum, maximum, and mean frequency that the sequence appears in a list of samples. For TCRB sequences, the prevalence (%) and the published antigen specificity of that sequence are also provided.

```
top.freq <- topFreq(productive.aa = productive.aa, percent = 0.1)
head(top.freq)
```

```
                aminoAcid minFrequency maxFrequency meanFrequency
373        CASSPPTGERDTQYF    0.4067472    17.798977     8.0931668
412 CASSQDLMTVDSLFAGANVLTF    1.5512222    10.542803     5.6504197
418        CASSQDRTGQYGYTF    0.4278417     1.088690     0.6955508
342        CASSPAGAYYNEQFF    0.3955412    19.139614    11.6126951
296          CASSLQGREKLFF    4.0790673    13.821673     7.3429132
556       CASSWPGLASFNEQFF    0.2489508     5.405405     1.0181321
    numberSamples prevalence antigen
373             9        1.8
412             9        1.8
418             9        3.6
342             8        5.5
296             8       30.9
556             8        1.8
```

One very useful thing to do is merge the output of `seqMatrix` and `topFreq`.

```
top.freq <- topFreq(productive.aa = productive.aa, percent = 0)
top.freq.matrix <- merge(top.freq, sequence.matrix)
head(top.freq.matrix)[1:12]
```

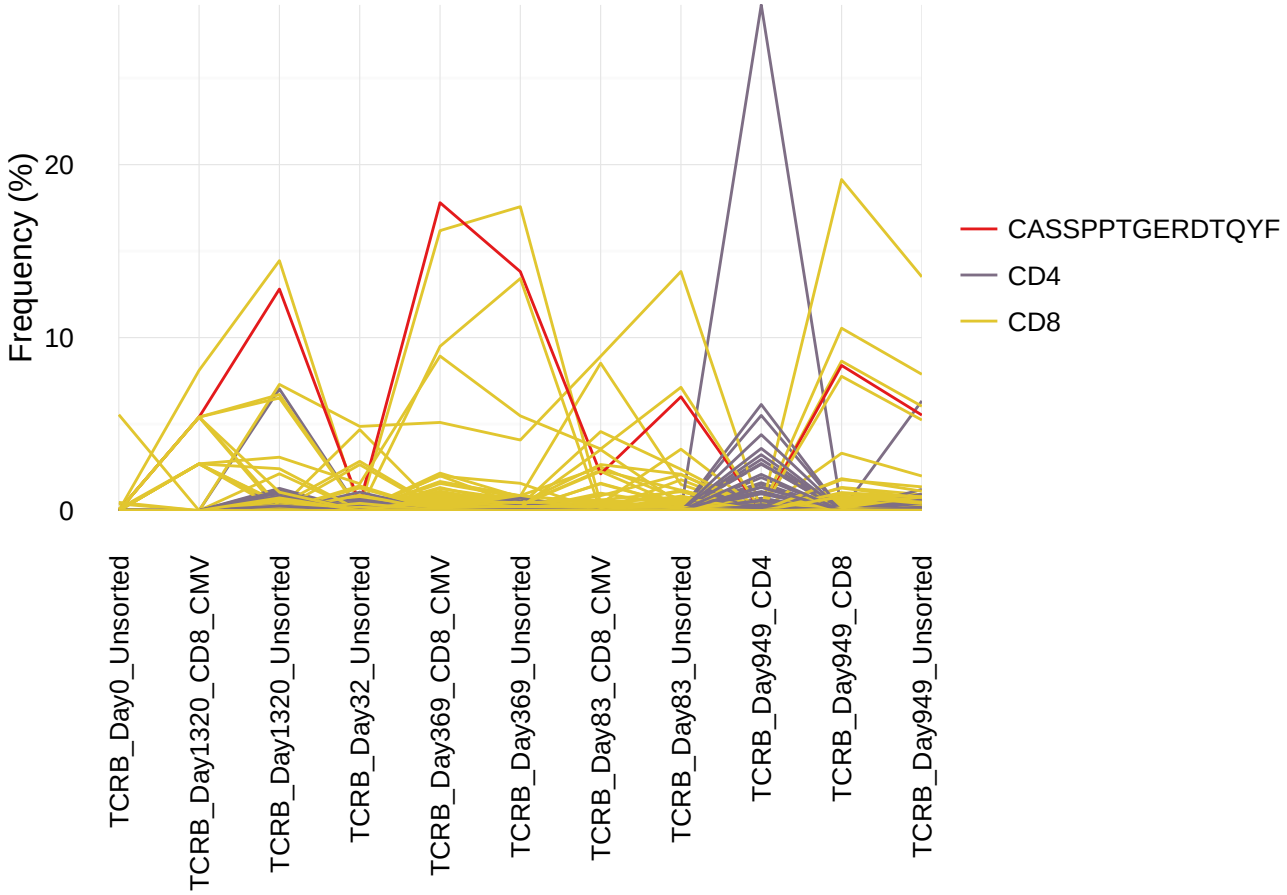
```
          aminoAcid numberSamples minFrequency maxFrequency meanFrequency
1     CAAGDTTLYEQYF             1   0.01799961   0.01799961    0.01799961
2      CAAGRDLNIQYF             1   0.03950662   0.03950662    0.03950662
3      CAAGTSGDTQYF             1   0.01830743   0.01830743    0.01830743
4    CAARGGGESYEQYF             1   0.03654798   0.03654798    0.03654798
5 CAATRRQGDVMNTEAFF             1   0.03771095   0.03771095    0.03771095
6    CACSRDRGSDTQYF             1   0.01957292   0.01957292    0.01957292
  prevalence antigen TCRB_Day0_Unsorted TCRB_Day1320_CD8_CMV
1        0.0                 0.01799961                    0
2        0.0                 0.00000000                    0
3        1.8                 0.00000000                    0
4        0.0                 0.03654798                    0
5        0.0                 0.00000000                    0
6        0.0                 0.00000000                    0
  TCRB_Day1320_Unsorted TCRB_Day32_Unsorted TCRB_Day369_CD8_CMV
1                     0                   0                   0
2                     0                   0                   0
3                     0                   0                   0
4                     0                   0                   0
5                     0                   0                   0
6                     0                   0                   0
```

Tracking sequences across samples

To visually track the frequency of sequences across multiple samples, use the function `cloneTrack`. This function takes the output from the `seqMatrix` function. You can specify a character vector of amino acid sequences using the parameter `track`

to highlight those sequences with a different color. Alternatively, you can highlight all of the sequences from a given sample using the parameter `map`. If the mapping feature is use, then you must specify a productive amino acid list and a character vector of labels to title the mapped samples. To hide sequences that are not being tracked or mapped, set `unassigned` to FALSE.

```
cloneTrack(sequence.matrix = sequence.matrix,
           productive.aa = productive.aa,
           map = c("TCRB_Day949_CD4", "TCRB_Day949_CD8"),
           label = c("CD4", "CD8"),
           track = "CASSPPTGERDTQYF",
           unassigned = FALSE)
```



Refer to the `cloneTrack` manual for examples on how to reformat the chart using ggplot2 function.

Comparing V(D)J gene usage

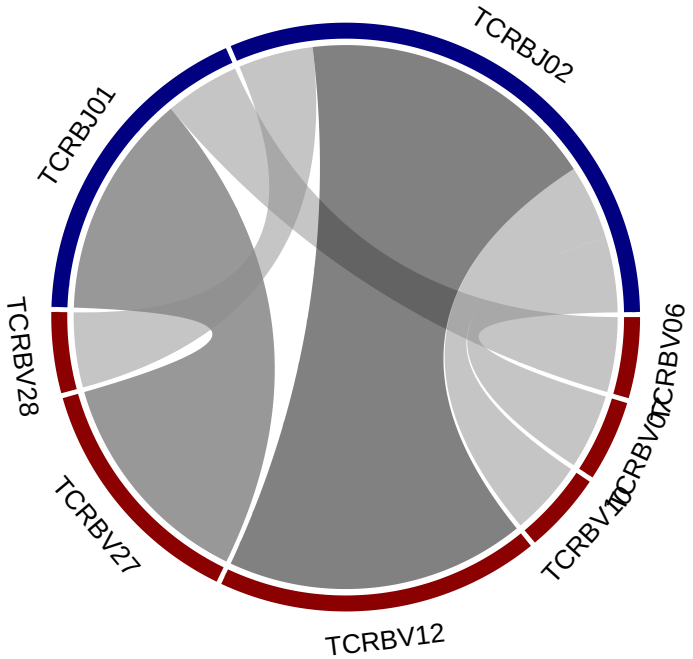
To compare the V, D, and J gene usage across samples, start by creating a data frame of V, D, and J gene counts and frequencies using the function `geneFreq`. You can specify if you are interested in the "VDJ", "DJ", "VJ", "DJ", "V", "D", or "J" loci using the `locus` parameter. Set `family` to TRUE if you prefer the family names instead of the gene names as reported by ImmunoSeq.

```
vGenes <- geneFreq(productive.nt = productive.nt, locus = "V", family = TRUE)
head(vGenes)
```

```
                samples familyName  count frequencyGene
1 TCRB_Day949_Unsorted    TCRBV02  16707      1.409503
2 TCRB_Day949_Unsorted    TCRBV03  44147      3.724508
3 TCRB_Day949_Unsorted    TCRBV04 176436     14.885207
4 TCRB_Day949_Unsorted    TCRBV05  89025      7.510687
5 TCRB_Day949_Unsorted    TCRBV06  97462      8.222483
6 TCRB_Day949_Unsorted    TCRBV07 106095      8.950815
```

To create a chord diagram showing VJ or DJ gene associations from one or more more samples, combine the output of `geneFreq` with the function `chordDiagramVDJ`. This function works well the `topSeqs` function that creates a data frame of a selected number of top productive sequences. In the example below, a chord diagram is made showing the association between V and J genes of just the single dominant clones in each sample. The size of the ribbons connecting VJ genes correspond to the number of samples that have that recombination event. The thicker the ribbon, the higher the frequency of the recombination.

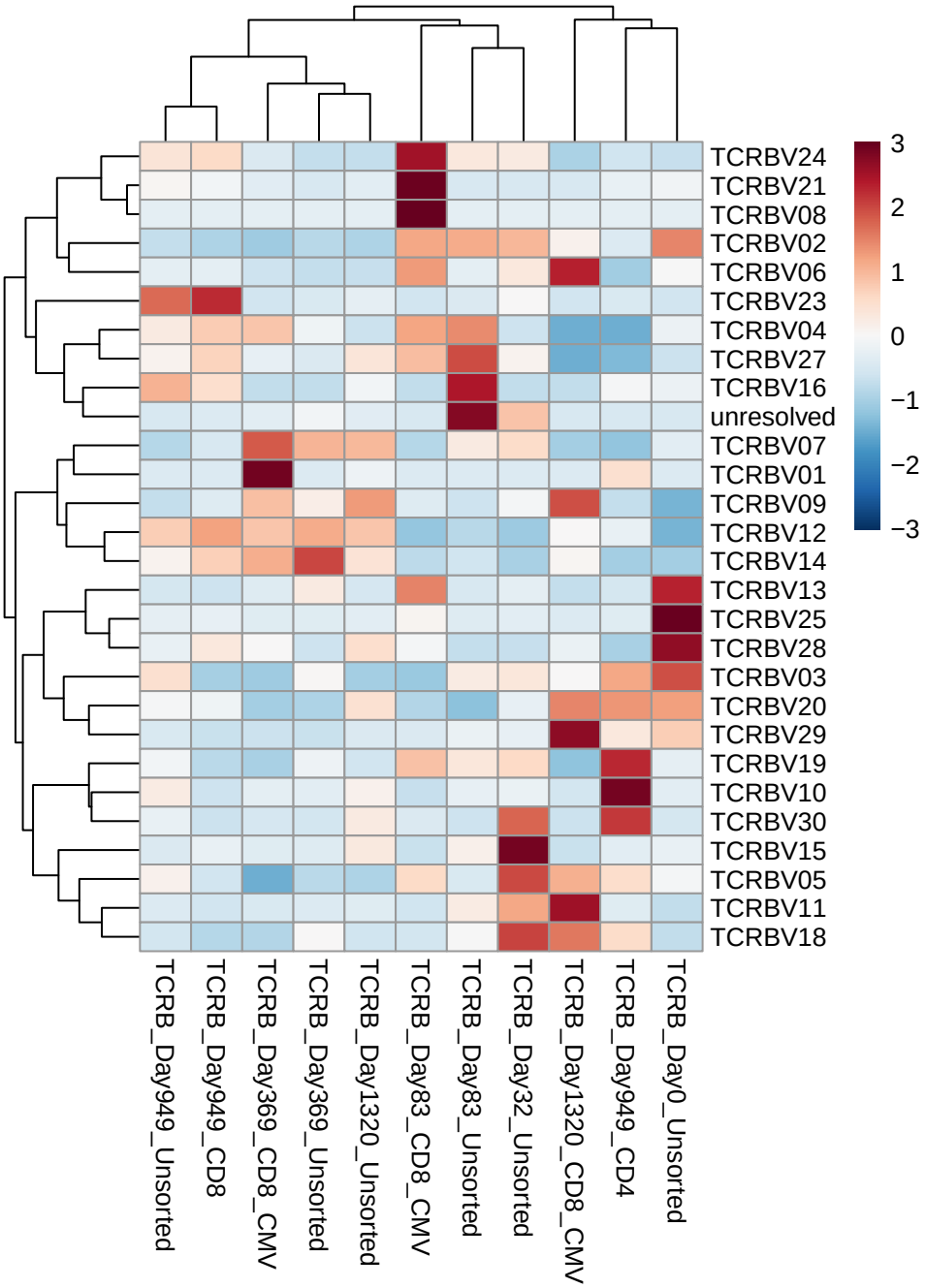
```
top.seqs <- topSeqs(productive.seqs = productive.nt, top = 1)
chordDiagramVDJ(sample = top.seqs,
                association = "VJ",
                colors = c("darkred", "navyblue"))
```



You can also visualize the results of `geneFreq` as a heat map, word cloud, our cumulative frequency bar plot with the support of additional R packages as shown below.

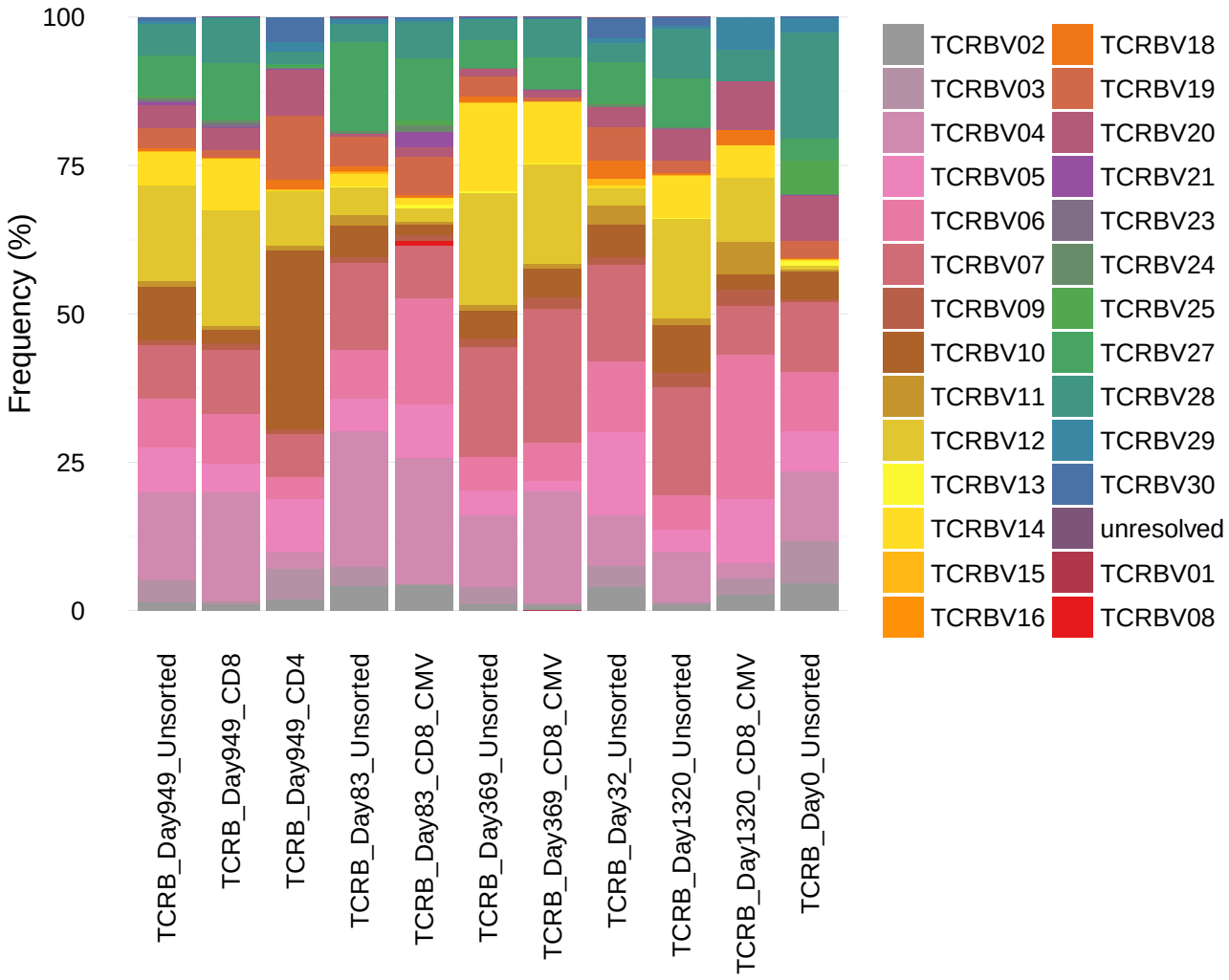
```
vGenes <- geneFreq(productive.nt = productive.nt, locus = "V", family = TRUE)
library(RColorBrewer)
library(grDevices)
RedBlue <- grDevices::colorRampPalette(rev(RColorBrewer::brewer.pal(11, "RdBu")))(256)
library(wordcloud)
wordcloud::wordcloud(words = vGenes[vGenes$samples == "TCRB_Day83_Unsorted", "familyName"],
                     freq = vGenes[vGenes$samples == "TCRB_Day83_Unsorted", "frequencyGene"],
                     colors = RedBlue)
```

```
library(reshape)
vGenes <- reshape::cast(vGenes, familyName ~ samples, value = "frequencyGene", sum)
rownames(vGenes) = as.character(vGenes$familyName)
vGenes$familyName = NULL
library(pheatmap)
pheatmap::pheatmap(vGenes, color = RedBlue, scale = "row")
```




```
vGenes <- geneFreq(productive.nt = productive.nt, locus = "V", family = TRUE)
library(ggplot2)
multicolors <- grDevices::colorRampPalette(rev(RColorBrewer::brewer.pal(9, "Set1")))(28)
ggplot2::ggplot(vGenes, aes(x = samples, y = frequencyGene, fill = familyName)) +
  geom_bar(stat = "identity") +
  theme_minimal() +
  scale_y_continuous(expand = c(0, 0)) +
  guides(fill = guide_legend(ncol = 2)) +
  scale_fill_manual(values = multicolors) +
  labs(y = "Frequency (%)", x = "", fill = "") +
```

```
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust = 1))
```



Removing sequences from the dataset

Occasionally you may identify one or more sequences in your data set that appear to be contamination. You can remove an amino acid sequence from all data frames using the function `removeSeq` and recompute frequencyCount for all remaining sequences.

```
searchSeq(list = productive.aa, sequence = "CASSDLIGNGKLFF")
```

```
                sample      aminoAcid count frequencyCount
49  TCRB_Day0_Unsorted CASSDLIGNGKLFF  3494    0.383479434
668    TCRB_Day949_CD8 CASSDLIGNGKLFF    82    0.004444141
    estimatedNumberGenomes
49                      55
668                      1
```

```
cleansed <- removeSeq(file.list = productive.aa, sequence = "CASSDLIGNGKLFF")
searchSeq(list = cleansed, sequence = "CASSDLIGNGKLFF")
```

```
No sequences found.
```

Merging samples

If you need to combine multiple samples into one, use the `mergeFiles` function. It merges two or more sample data frames into a single data frame and aggregates count, frequencyCount, and estimatedNumberGenomes.

```
TCRB_Day949_Merged <- mergeFiles(samples = c("TCRB_Day949_CD4", "TCRB_Day949_CD8"),
                                 file.list = file.list)
```

Conclusion

Advances in high-throughput sequencing have enabled characterizing T and B lymphocyte repertoires with unprecedented depth. LymphoSeq was developed as a tool to assist in the analysis of targeted next generation sequencing of the hypervariable CDR3 region of T and B cell receptors. The three key features of this R package are to characterize lymphocyte repertoire diversity, compare two or more lymphocyte repertoires, and track the frequency of CDR3 sequences across multiple samples. LymphoSeq also provides the unique ability to search for sequences in a curated database of published TCRB sequences with known antigen specificity. Finally, LymphoSeq can assign the percent prevalence that any given TCRB sequence appears in a the peripheral blood in healthy population of donors.

Session info

```
sessionInfo()
```

```
## R version 3.3.1 (2016-06-21)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 16.04.1 LTS
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=C
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] ggplot2_2.1.0      pheatmap_1.0.8     reshape_0.8.5
## [4] wordcloud_2.5      RColorBrewer_1.1-2 LymphoSeq_1.2.0
## [7] LymphoSeqDB_0.99.2
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.7          knitr_1.14           magrittr_1.5
##  [4] munsell_0.4.3        colorspace_1.2-7     R6_2.2.0
##  [7] ineq_0.2-13          dplyr_0.5.0          stringr_1.1.0
## [10] plyr_1.8.4           tools_3.3.1          grid_3.3.1
## [13] data.table_1.9.6     gtable_0.2.0         circlize_0.3.9
## [16] DBI_0.5-1            lambda.r_1.1.9       futile.logger_1.4.3
## [19] htmltools_0.3.5      lazyeval_0.2.0       yaml_2.1.13
## [22] assertthat_0.1       digest_0.6.10        tibble_1.2
## [25] formatR_1.4          GlobalOptions_0.0.10 futile.options_1.0.0
## [28] slam_0.1-38          shape_1.4.2          VennDiagram_1.6.17
## [31] evaluate_0.10        rmarkdown_1.1        labeling_0.3
## [34] stringi_1.1.2        scales_0.4.0         chron_2.3-47
```