Exploring CD8+ T-cell Specificities using Single Cell Immune Profiling with an Outlook to Reproducible Bio Data Science

William Hagedorn-Rasmussen

Table of contents

1 Imports

```
library(TCRSequenceFunctions)
library(gt)
library(notly)
```

2 Introduction

This package, TCRSequenceFunctions, is a collection of functions made for working with data sets from a Single Cell Immune Profiling experiment made by 10x Genomics [1]. There is a total of four data sets which all follow the same general structure. They differ in that, they contain data from each their own respective donor.

The data sets contains binding counts between the donors' library of T-Cell Receptors (TCRs) and a set of peptide-major histocompatibility complexes (pMHCs). The before-mentioned binding counts are so called unique molecular identifier (UMI) counts. For an explanation of all columns see Section 6.

In the following sections, each of the functions contained in TCRSequenceFunctions will be explained, demonstrated and reasoned for. Generally, they can be divided into three types: Cleaning, Augmenting and Modelling where the main goal of the two first is to make the data tidy.

Lastly will be a short section on a Shiny Package, TCRSequenceShiny, which utilizes these functions to make a user-friendly interactive interface for data exploration.

3 Tidying the data

The aim of tidying the data is to enable the data handling, and to ensure a reproducible result. Firstly, the data is cleaned e.g. by making sure, all cells only contain one piece of information. Afterwards, some augmented was needed to enable the modelling. This was done by e.g. adding new columns. A wrapper function was used to run all the preparation functions: `run_all_prep()`. This wrapper simply takes one of the raw data files included in the package as input, and pipe it through all the preparation functions, and output tidy data as in Listing 1.

Listing 1 How to pipe data through wrapper

```
data_donor_one_raw %>%
  run_all_prep()
```

3.1 Cleaning

As mentioned above, cleaning the data is mostly focusing on handling already present data and/or re-structure the data frame. The list of cleaning functions are as follows:

1. `remove_unnecessary_columns()`
2. `find_non_promiscuous_split_TCR_sequences()`
3. `pivot_longer_TCR_sequences()`
4. `add_chain_ident_remove_prefix()`
5. `pivot_longer_pMHC()`
6. `tidy_pMHC_names()`

The first function takes the raw data frame as input, and simply removes the unnecessary columns as these aren't needed. By default, the columns removed are those containing `"_binder"` and the column `"cell_clono_cdr3_nt"`.

`find_non_promiscuous_split_TCR_sequences()`, `pivot_longer_TCR_sequence()` and `add_chain_ident_remove_prefix()` works in close relation with one another. The main purpose is to tidy the TCR-sequences, as to not have cells with multiple pieces of information. Table 1 shows three examples as to how these are written. A column is added to indicate a non-promiscuous pair (i.e., a pair with one alpha- and beta chain respectively) and contains the TCR-sequence of said pair. This is done since the sequences of non-promiscuous pairs are needed for modelling in `relevant_binder_frequency_plot()`. The TCR-sequences are then split into new columns, one for each chain. These new columns are then pivot longer into the column `TCR_sequence` by `pivot_longer_TCR_sequence()`. Lastly, a new column is added to indicate chain type, `chain`, and the chain indicator `"TR[A|B]"` is removed. The result can be seen in Table 2.

Table 1: *A snippet of data_donor_one_raw to show two examples as to how TCR-sequences are written*

cell_clono_cdr3_aa
TRA:CAASVSIWTGTASKLTF;TRA:CAAWDMEYGNKLVF;TRB:CAISDPGLAGGGGEQFF
TRB:CASDTPVGQFF
TRA:CASYTDKLIF;TRB:CASSGGSISTDTQYF

Table 2: *A snippet of the output from find_non_promiscuous_split_TCR_sequences() to show a tidy version of the TCR-sequences*

TCR_sequence	chain	non_promiscuous_pair
CAASVSIWTGTASKLTF	alpha	NA
CAAWDMEYGNKLVF	alpha	NA
CAISDPGLAGGGGEQFF	beta	NA
CASDTPVGQFF	beta	NA
CASYTDKLIF	alpha	CASYTDKLIF;CASSGGSISTDTQYF
CASSGGSISTDTQYF	beta	CASYTDKLIF;CASSGGSISTDTQYF

A similar process is applied to all column names which follow the general structure: `"allele_peptide_peptidesource"`. This is done through `pivot_longer_pMHC()` and `tidy_pMHC_names()`. First the names are pivot longer with their associated UMI-counts as values. Here, only non-zero values are kept as they are otherwise irrelevant. The names, now as rows, are made tidy by splitting into three new columns:

1. `allele`
2. `peptide`
3. `peptide_source`

This step is important as to not have multiple pieces of information in one cell, and as they are used for modelling in `summarise_with_filter()`.

3.2 Augmenting

Generally speaking, augmenting the data refers to adding new columns which enable later modelling. In some cases, it makes more sense to do augmenting while cleaning (e.g. `find_non_promiscuous_split_TCR_sequences()`). The list of augmenting functions are as follows:

1. `add_max_non_specific_binder()`
2. `evaluate_binder()`
3. `add_TCR_combination_identifier()`

The first and second function are both required to find the relevant binders. A relevant binder has four requirements which needs to be met:

1. UMI-count needs to be larger than some threshold
2. The UMI-count has to be another threshold times higher than the non-specific binder with the highest UMI-count for that barcode

3. If multiple specificities exist, only use the one with the highest UMI-count
4. Disregard a cell if it shows specificity towards more than four pMHC

And so, the purpose of the first function is to find the highest UMI-count of all non-specific binders for each barcode. The count is added in a new column called `max_non_specific_binder`. The second function then uses the above mentioned requirements to evaluate all binding events. The evaluation is used in e.g. `relevant_binder_frequency_plot()`. Some examples of evaluation can be seen in Table 3.

Table 3: *Examples of how different combinations of UMI-count and max_non_specific_binder are evaluated*

	UMI_count	**max_non_specific_binder**
FALSE		
A0201_KLQCVDLHV_PSA146-154	1	1
A1101_AVFDRKSDAK_EBNA-3B_EBV	2	1
A0201_KLQCVDLHV_PSA146-154	1	1
TRUE		
A1101_AVFDRKSDAK_EBNA-3B_EBV	451	0
A1101_AVFDRKSDAK_EBNA-3B_EBV	451	0
A1101_AVFDRKSDAK_EBNA-3B_EBV	451	0

Lastly, `add_TCR_combination_identifier()` categorizes the TCR-sequence chain combinations for each barcode. I.e. it determines which of the following groups the TCR-sequence for a barcode belongs to:

1. Containing only one alpha chain and no beta chains
2. Containing no alpha chain and only one beta chain
3. Containing one alpha chain and beta chain respectively
4. Some other combination

This allows for a distribution of the above-mentioned categories to perform a quick check on the data through `alpha_beta_pair_distributions()`. A few examples can be seen in Table 4.

Table 4: *Examples of how barcodes are categorized depending on their alpha- and beta chain combination*

barcode	**chain**	**TCR_combination**
AAACCTGAGACAAAGG-4	alpha	other
AAACCTGAGACAAAGG-4	alpha	other
AAACCTGAGACAAAGG-4	alpha	other
AAACCTGAGACAAAGG-4	alpha	other
AAACCTGAGACAAAGG-4	beta	other

Table 5: *Dimensions of data sets for* ***(a)*** *Donor 1,* ***(b)*** *Donor 2,* ***(c)*** *Donor 3 and* ***(d)*** *Donor 4 before and after being prepared by the wrapper* `run_all_prep()`

(a)

Donor 1	raw	tidy
Number of columns	118	27
Number of rows	46526	512264

(b)

Donor 2	raw	tidy
Number of columns	118	27
Number of rows	77854	860757

(c)

Donor 3	raw	tidy
Number of columns	118	27
Number of rows	37824	581484

(d)

Donor 4	raw	tidy
Number of columns	118	27
Number of rows	27308	190482

AAACCTGAGACAAAGG-4	beta	other
AAACCTGAGACTGTAA-34	beta	one_beta_only
AAACCTGAGACTGTAA-34	beta	one_beta_only
AAACCTGAGACTGTAA-34	beta	one_beta_only
AAACCTGAGAGCCCAA-5	alpha	one_alpha_one_beta
AAACCTGAGAGCCCAA-5	alpha	one_alpha_one_beta

3.3 Change of dimensions

Naturally, the dimensions of all included data sets change after being piped through `run_all_prep()`. Table 5 shows the dimensions for both the raw and tidy data sets stratified on donor. The number of columns are always reduced to 27, but the number of rows depend on the original number of rows and the data itself.

4 Modelling

All of the tidying done in the above enables the modelling which are described in this section. For each model, a subsection is included to describe the purpose and the output of said models. The list of modeling functions are:

1. `summarise_with_filter()`
2. `relevant_binder_frequency_plot()`
3. `alpha_beta_pair_distributions()`
4. `alpha_beta_consistency()`

4.1 `summarise_with_filter()`

As input, this function takes a tidy data frame. For each cell - barcode - it counts all of the relevant binding events stratified on a user input. Through its arguments it's possible to decide the stratification as shown in Table 6. From the output it is then apparent, that some alleles and peptides bind more often than others which could prove interesting.

```
data_donor_one_tidy %>%
  summarise_with_filter(summarise_by = c("allele", "peptide"))
```

Table 6: *Output of the function summarise_with_filter(). For each allele and for each peptide a count as noted indicating the number of binding events to cells.*

allele	peptide	count
A0201	CLLWSFQTSA	1
A0201	ELAGIGILTV	242
A0201	FLASKIGRLV	8
A0201	FLYALALLL	21
A0201	GILGFVFTL	2617
A0201	GLCTLVAML	39
A0201	IMDQVPFSV	5
A0201	KTWGQYWQV	7
A0201	KVLEYVIKV	3
A0201	LLDFVRFMGV	142

4.2 `relevant_binder_frequency_plot()`

As the function above, this one also takes a tidy data frame as input. The model counts the number of pMHC which bind to non-promiscuous pairs stratified on barcode. Based om that count, a frequency is calculated showing how often a pMHC bind to a non-promiscuous pair out of all the other pMHC which bind to that exact pair. The output is then a plotly plot (i.e., its interactive) where each dot represent a binding event between pMHC and a non-promiscuous pair. The size of the dot increases with increasing frequency. The axis ticks have been disables due to cluttering, but as mentioned the values can be found by hovering a dot. It is important to mention, that only interactions evaluated to relevant by `evaluate_binder()` is included. Hence, a change of threshold in that function will directly change the output of this function. Furthermore, it is possible to provide the function with a maximum frequency. If one is interested in only ambiguous sequences (i.e., TCR-sequences which bind multiple pMHC), the max frequency should be set to something less than `1`. For a demonstration on how to use the model see Figure 1 (will only work properly when rendering in HTML format).

```
data_donor_four_tidy %>%
  relevant_binder_frequency_plot()
```

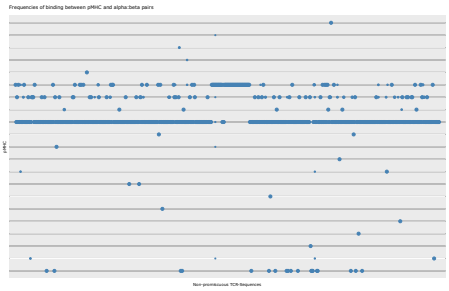

Figure 1: *Output of the function* `relevant_binder_frequency_plot`*. Each dot represent a relevant binding between a non-promiscous TCR-sequence and pMHC.*

4.3 `alpha_beta_pair_distribution()`

As with the other model functions, it takes a tidy data frame as input with the option to only look at distributions for a single pMHC provided through the arguments. From `add_TCR_combination_identifier()` we have the categorization of each cell. By making each barcode unique, and counting number of occurrences for each category, a frequency distribution is obtainable. The output is a bar plot showing exactly that (see Figure 2). It can be used as a quality check of the data.

```
data_donor_four_tidy %>%
  alpha_beta_pair_distribution()
```

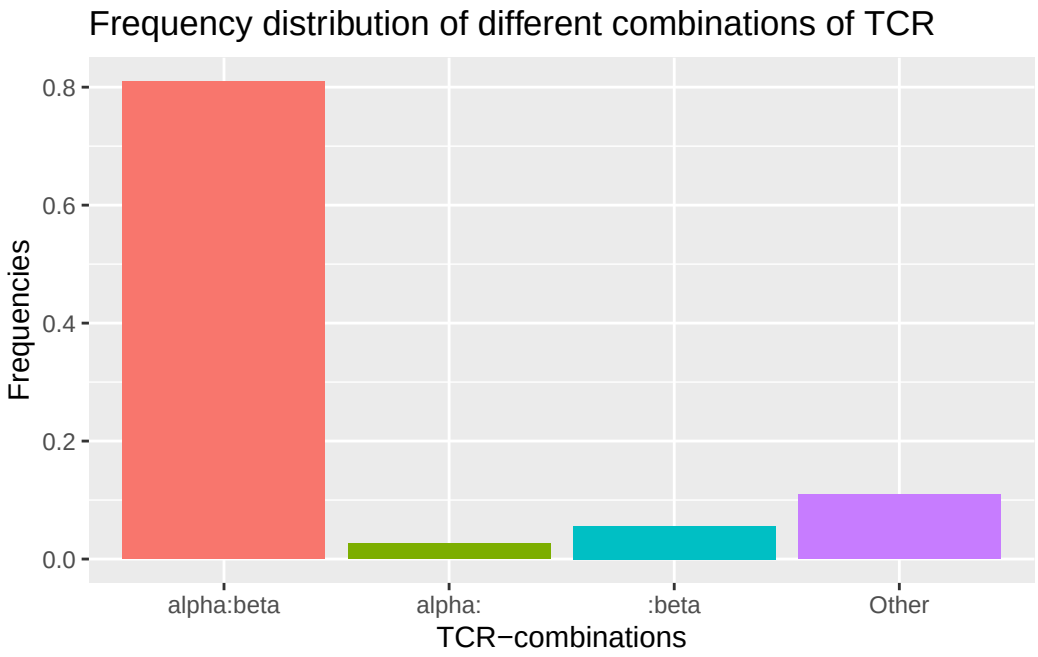

Figure 2: *Output of the function* `alpha_beta_pair_distribution`*. Each bar represent the freuqency of that respective category.*

4.4 `alpha_beta_consistency()`

The input to the function is a tidy data frame from which a frequency to plot is made. The function groups on the barcode, then make TCR-sequences unique, and ungroup again to do a data set wide counting of the number of TCR-sequences. A distinctiveness score is then calculated by comparing the number of completely unique sequences with the total number of TCR-sequences. I.e., a lower score means less distinctive. The calculations are done by stratifying on `chain`. This enables a quick quality check of the data set to whether it follows theory or not. The output is then a bar plot showing the distinctiveness score (See Figure 3).

```
data_donor_four_tidy %>%
  alpha_beta_consistency()
```

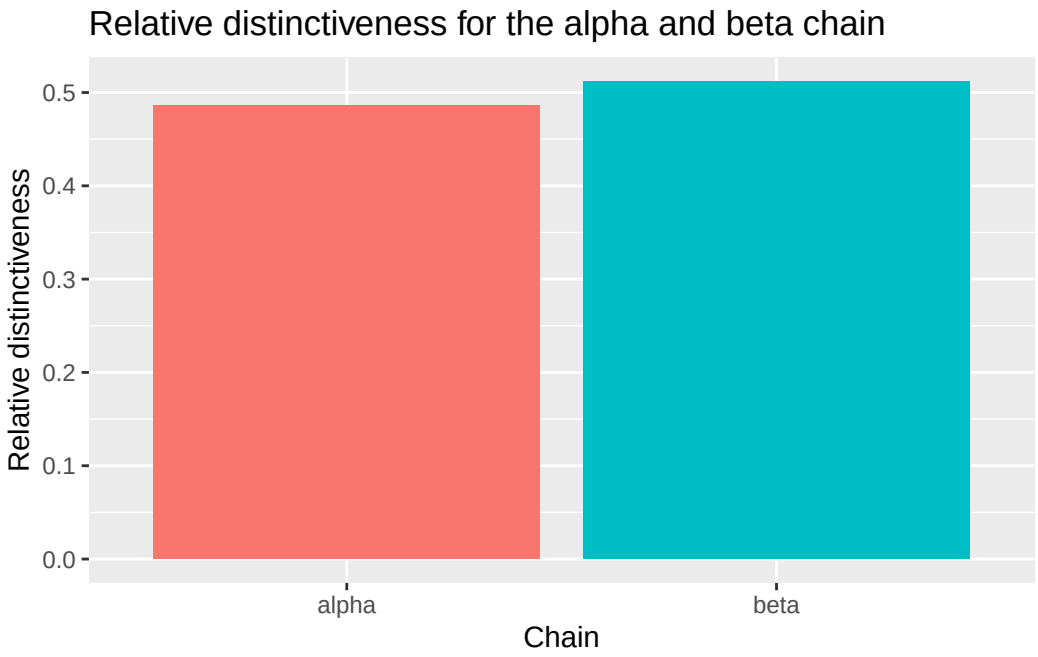

Figure 3: *Output of the function* `alpha_beta_consistency`*. Each bar represent the distinctiveness of that chain type.*

5 Shiny Integration

References

6 Appendix

Appendix A

[1] 10X Genomics. 2022. A New Way of Exploring Immunity - Linking Highly Multiplexed Antigen Recognition to Immune Repertoire and Phenotype. *10x Genomics* (2022), 1–13. Retrieved from https://www.10xgenomics.com/resources/document-library/a14cde