Introduction to PepSeq

Derek Sonderegger

2019-07-15

PepSeq is a package developed to facilitate the exploration of PMI’s PepSeq data. Given a pulldown, we want to explore how the cleaved-uncleaved signal varies across different locations in the proteins.

To begin, we will install the package via the following R commands:

# install PepSeq package from Github. This is only necessary when upgrading to a new version  
library(devtools)  
install\_github('dereksonderegger/PepSeq') # install latest version of Derek's PepSeq tools

library(tidyverse) # load the usual set of data processing tools  
library(PepSeq) # load Derek's library of routines

The first step is to set the working directory to wherever the pulldown .csv file is located. This can be done via the point and click interface (Session -> Set Working Directory) or using the command setwd. Once the working directory is set to where ever the data file lives, you can define the input file.

setwd('~/GitHub/PepSeq/inst/extdata')  
input\_file <- '2019\_06\_25\_Pulldown.csv'

Alternatively, you could just list the full pathname to the input and output files and ignore the working directory business.

input\_file <- '~/GitHub/PepSeq/inst/extdata/2019\_05\_10\_Pulldown.csv'  
# input\_file <- '~/GitHub/PepSeq/data-raw/2019\_05\_10\_Pulldown.csv'

PepSeq assumes that a pulldown .csv file is arrange in a format with one protein/location combination per line and then several sets of cleaved/uncleaved pairs and possibly columns of data that aren’t cleaved/uncleaved but rather just some other signal (e.g. output from another model as to the likelihood of binding). There are also several columns that define meta data that we don’t need in our program, but will need to include in subsequent output.

read.csv(input\_file) %>% colnames()  
#> [1] "library\_member"   
#> [2] "Class"   
#> [3] "Seq....Rank.of.sequence.based.on.ratio.of.ligands.per.seq.length."   
#> [4] "protein\_ID"   
#> [5] "Peptide.start"   
#> [6] "Peptide.end"   
#> [7] "Peptide.seq"   
#> [8] "X.ve..ve.set"   
#> [9] "X..within.set"   
#> [10] "Seq....Rank.of.sequence.based.on.ratio.of.ligands.per.seq.length..1"  
#> [11] "Accession"   
#> [12] "Organism.Name"   
#> [13] "Organism.ID"   
#> [14] "Protein.name"   
#> [15] "X15.mer.peptide.start"   
#> [16] "X15.mer.peptide\_end"   
#> [17] "Different.protein."   
#> [18] "Old\_ID"   
#> [19] "Type"   
#> [20] "Cl\_DRB1.14.01"   
#> [21] "Un\_DRB1.14.01"   
#> [22] "Cl\_DRB1.15.03"   
#> [23] "Un\_DRB1.15.03"   
#> [24] "Cl\_DRB1.15.02"   
#> [25] "Un\_DRB1.15.02"   
#> [26] "Cl\_DRB1.14.03"   
#> [27] "Un\_DRB1.14.03"   
#> [28] "Cl\_15.01\_A"   
#> [29] "Un\_15.01\_A"   
#> [30] "Cl\_07.01\_A"   
#> [31] "Un\_07.01\_A"   
#> [32] "Cl\_11.01\_A"   
#> [33] "Un\_11.01\_B"   
#> [34] "Cl\_04.01\_A"   
#> [35] "Un\_04.01\_A"   
#> [36] "Cl\_13.02\_A"   
#> [37] "Un\_13.02\_A"   
#> [38] "Cl\_01.01\_C"   
#> [39] "Un\_01.01\_C"   
#> [40] "Cl\_03.01\_A"   
#> [41] "Un\_03.01\_A"   
#> [42] "Cl\_04.05\_A"   
#> [43] "Un\_04.05\_A"

The orderings of all the columns doesn’t matter, but you need to be able to identify the protein and position columns and each of the cleaved/uncleaved pairs need to some sort of cleaved/uncleaved prefix or postfix. For example, columns might have a group name followed by an ‘*Cleaved’ or ’Uncleaved’. Alternatively, the column names might begin with ’Cl’ or ’Un*’. So long as the Cleaved\_Type\_Indicators denotes them as a vector strings (order is cleaved then uncleaved), then the function will search for those two strings within the column name and appropriately identify the cleaved and uncleaved values. If a column name doesn’t contain the ‘Cleaved’ or ‘Uncleaved’ suffix on a column then we assume the column is a measurement and we will directly pass the values through without separating the cleaved/uncleaved and doing any standardization.

In order to allow there to be many other interesting columns that should be ignored in the visualization, PepSeq requires you to indicate which columns are to be plotted by either having an initial suffix indicating it is data or to indicate the responses by column location (e.g. columns 3 to 6).

# There is a function to import from a .csv data file  
data <- import\_pulldown(  
 file = input\_file,   
 standardization\_method = 'additive',  
 protein\_column = 'Accession',   
 position\_column = 'Peptide.start',  
 read\_indicator = 20:42, # Which columns are data  
 Cleaved\_Type\_Indicators = c('Cl\_', 'Un\_') ) # cleaved/uncleaved columns end with \_Cleaved or \_Uncleaved  
#> Warning in max(df$signal, na.rm = TRUE): no non-missing arguments to max;  
#> returning -Inf  
  
#> Warning in max(df$signal, na.rm = TRUE): no non-missing arguments to max;  
#> returning -Inf  
  
#> Warning in max(df$signal, na.rm = TRUE): no non-missing arguments to max;  
#> returning -Inf

What is this thing we’ve just read in?

head(data) # Lets look at the columns in the resulting data frame  
#> # A tibble: 6 x 7  
#> # Groups: Group [1]  
#> index protein\_ID position Group cleaved uncleaved signal  
#> <int> <fct> <int> <chr> <int> <int> <dbl>  
#> 1 1 1PUO\_A 1 DRB1.14.01 NA 3 NA  
#> 2 2 1PUO\_A 2 DRB1.14.01 NA 4 NA  
#> 3 3 1PUO\_A 3 DRB1.14.01 NA 14 NA  
#> 4 4 1PUO\_A 4 DRB1.14.01 NA 11 NA  
#> 5 5 1PUO\_A 5 DRB1.14.01 NA 8 NA  
#> 6 6 1PUO\_A 6 DRB1.14.01 NA 4 NA

The protein\_ID and position came directly from the input file. The cleaved/uncleaved columns are the raw counts from the imput file with no adjusting for read depth. The signal column is calculated from the cleaved/uncleaved columns after accounting for read depth.

The import function produces a data frame where the “Group” column denotes the allele group we are working with.

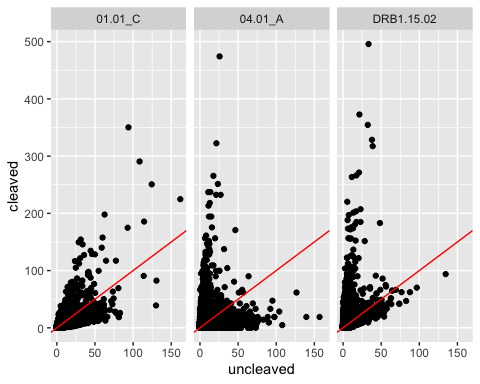
# The data file likely contains columns that are useful for identification of proteins. Or perhaps   
# the raw peptide sequence. So the `import\_pulldown\_metadata` function allows us to pick out  
# the meta data columns that you want.  
Meta\_data <- import\_pulldown\_metadata(  
 file = input\_file,   
 protein\_column = 'protein\_ID',   
 position\_column = 'Peptide.start',  
 meta\_columns = c('library\_member', 'Peptide.seq'))   
  
# By default the meta data is coming out as a factor. I want them to be character strings.  
Meta\_data <- Meta\_data %>%  
 mutate(library\_member = as.character(library\_member),  
 Peptide.seq = as.character(Peptide.seq) )

Notice that in each of the calls, we also had to specify which column specified the protein name and which column denotes the position along the protein.

# Simple Analyses

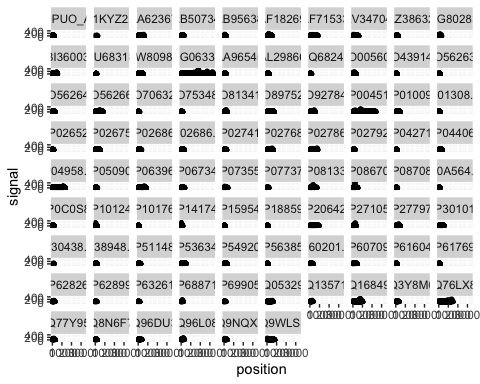
First we consider how to make a simple scatter plot of the cleaved vs uncleaved counts for each allele. Because the cleaved/uncleaved columns in the data frame are not standardized to account for the read depth,

# ggplot( data, aes(x=log(uncleaved), y=log(cleaved) ) ) +  
data %>%   
 group\_by( Group ) %>% # Do the standardization only across MHC experiments  
 mutate( uncleaved = uncleaved/sum(uncleaved) \* 1e5, # standardize the read depths...  
 cleaved = cleaved/sum(cleaved) \* 1e5 ) %>% # reads per 10,000  
 filter( Group %in% c('01.01\_C', 'DRB1.15.02', '04.01\_A') ) %>% # if you only want one or more group  
ggplot( ., aes(x=uncleaved, y=cleaved ) ) +  
 geom\_point() +   
 geom\_abline( intercept = 0, slope = 1, color='red') +  
 facet\_wrap( ~ Group )



I want to show how to graph the standardized signal across proteins.

data %>%  
 filter( Group %in% c('04.01\_A') ) %>% # Only do the 04.01\_A run, include all Proteins!  
ggplot( ., aes(x = position, y=signal )) +  
 geom\_point() +  
 facet\_wrap( ~ protein\_ID )



data %>%  
 filter( Group %in% c('01.01\_C', '04.01\_A') ) %>%  
 filter( protein\_ID %in% c('AAZ38632.1', 'ABG80283.1') ) %>%  
plot\_pulldown\_Shiny(input=., height=600,   
 peaks = TRUE, peak\_method = 'PoT', peak\_param = 10)  
  
data %>%  
 filter( Group %in% c('01.01\_C', '04.01\_A') ) %>%  
 filter( protein\_ID %in% c('AAZ38632.1', 'ABG80283.1') ) %>%  
 plot\_pulldown(input=., peaks=TRUE, peak\_param=10)

# Calculating the peaks

There is one main user function in the PepSeq package used for calculating the peaks:

data %>%  
 filter( Group %in% c('01.01\_C', '04.01\_A') ) %>%  
 filter( protein\_ID %in% c('AAZ38632.1', 'ABG80283.1') ) %>%  
identify\_peaks(., method='PoT', param = NULL)  
#> Using default thresholds  
#> # A tibble: 24 x 5  
#> # Groups: Group, protein\_ID [4]  
#> Group protein\_ID Peak Start End  
#> <chr> <fct> <int> <int> <int>  
#> 1 01.01\_C AAZ38632.1 1 32 37  
#> 2 01.01\_C AAZ38632.1 2 45 47  
#> 3 01.01\_C AAZ38632.1 3 72 77  
#> 4 01.01\_C AAZ38632.1 4 103 105  
#> 5 01.01\_C AAZ38632.1 5 108 112  
#> 6 01.01\_C AAZ38632.1 6 153 160  
#> 7 01.01\_C ABG80283.1 1 66 69  
#> 8 01.01\_C ABG80283.1 2 72 74  
#> 9 01.01\_C ABG80283.1 3 111 113  
#> 10 01.01\_C ABG80283.1 4 126 132  
#> # … with 14 more rows

Next we want to take the peak information and include the meta information so as to figure out the full peptide sequence corresponding to the peak.

peaks <- data %>%  
 filter( Group %in% c('01.01\_C', '04.01\_A') ) %>%  
 filter( protein\_ID %in% c('AAZ38632.1', 'ABG80283.1') ) %>%  
 identify\_peaks(., method='PoT', param = NULL)  
#> Using default thresholds  
  
peaks %>%   
 group\_by(Group, protein\_ID, Peak) %>%  
 group\_modify( ~ tibble(position = .x$Start : .x$End ) ) %>%  
 left\_join(Meta\_data)  
#> Joining, by = c("protein\_ID", "position")  
#> Warning: Column `protein\_ID` joining factors with different levels,  
#> coercing to character vector  
#> # A tibble: 120 x 6  
#> # Groups: Group, protein\_ID, Peak [24]  
#> Group protein\_ID Peak position library\_member Peptide.seq   
#> <chr> <chr> <int> <int> <chr> <chr>   
#> 1 01.01\_C AAZ38632.1 1 32 IEDBv1\_25219 FLDRLRRDQKSLKGR  
#> 2 01.01\_C AAZ38632.1 1 33 IEDBv1\_25220 LDRLRRDQKSLKGRG  
#> 3 01.01\_C AAZ38632.1 1 34 IEDBv1\_25221 DRLRRDQKSLKGRGS  
#> 4 01.01\_C AAZ38632.1 1 35 IEDBv1\_25222 RLRRDQKSLKGRGST  
#> 5 01.01\_C AAZ38632.1 1 36 IEDBv1\_25223 LRRDQKSLKGRGSTL  
#> 6 01.01\_C AAZ38632.1 1 37 IEDBv1\_25224 RRDQKSLKGRGSTLG  
#> 7 01.01\_C AAZ38632.1 2 45 IEDBv1\_25232 GRGSTLGLNIETATC  
#> 8 01.01\_C AAZ38632.1 2 46 IEDBv1\_25233 RGSTLGLNIETATCV  
#> 9 01.01\_C AAZ38632.1 2 47 IEDBv1\_25234 GSTLGLNIETATCVG  
#> 10 01.01\_C AAZ38632.1 3 72 IEDBv1\_25259 ESDEAFKMTMASALA  
#> # … with 110 more rows

Next we want to take each of the sequences that compose the peak and isolate the peak core, which we will define as the peptides that are in the all of the sequences that define the peak. The way we’ll do that is to take the first peptide sequence and peal off however many peptides are in the length.

peaks %>%   
 group\_by(Group, protein\_ID, Peak) %>%  
 mutate( position = Start, length = End - Start ) %>%  
 left\_join(Meta\_data) %>%   
 mutate( Peptide.seq = str\_sub( Peptide.seq, length+1 ) )  
#> Joining, by = c("protein\_ID", "position")  
#> Warning: Column `protein\_ID` joining factors with different levels,  
#> coercing to character vector  
#> # A tibble: 24 x 9  
#> # Groups: Group, protein\_ID, Peak [24]  
#> Group protein\_ID Peak Start End position length library\_member  
#> <chr> <chr> <int> <int> <int> <int> <int> <chr>   
#> 1 01.0… AAZ38632.1 1 32 37 32 5 IEDBv1\_25219   
#> 2 01.0… AAZ38632.1 2 45 47 45 2 IEDBv1\_25232   
#> 3 01.0… AAZ38632.1 3 72 77 72 5 IEDBv1\_25259   
#> 4 01.0… AAZ38632.1 4 103 105 103 2 IEDBv1\_25290   
#> 5 01.0… AAZ38632.1 5 108 112 108 4 IEDBv1\_25295   
#> 6 01.0… AAZ38632.1 6 153 160 153 7 MBA\_B\_002131   
#> 7 01.0… ABG80283.1 1 66 69 66 3 MBA\_B\_000955   
#> 8 01.0… ABG80283.1 2 72 74 72 2 MBA\_B\_000961   
#> 9 01.0… ABG80283.1 3 111 113 111 2 IEDBv1\_17750   
#> 10 01.0… ABG80283.1 4 126 132 126 6 IEDBv1\_17764   
#> # … with 14 more rows, and 1 more variable: Peptide.seq <chr>