iCLIP project: U2AF65

Melina Klostermann

26 August, 2020

Contents

```r
library(GenomicRanges)
library(rtracklayer)
library(knitr)
library(GenomicFeatures)
library(dplyr)
library(ggpubr)
library(BSgenome.Hsapiens.UCSC.hg19)
#library(BSgenome.Hsapiens.NCBI.GRCh38)
library(biomaRt)
library(ggpointdensity)
```

```r
# reimport your Binding sites
path_to_BS_RData <- "path/to/your/BS/from/yesterday.RData"
```

```r
load(path_to_BS_RData)

# split up plus and minus Binding sites
Binding_sites_n_cl_plus <- Binding_sites_n_cl[strand(Binding_sites_n_cl)=="+"]
Binding_sites_n_cl_minus <- Binding_sites_n_cl[strand(Binding_sites_n_cl)=="-"]
```

1 Reproducibility of binding sites between replicates

For the binding site definition above, we used the merged signal of all replicates of our experiment. In order to account for their reproducibility between replicates, we check how many of the replicates support the detected binding sites. For this, we overlap the binding sites with the crosslinking signal of the individual replicates and sum up how many crosslink events from each replicates fell into each binding site. By setting a threshold for the minimum number of crosslink events (here: 1 crosslink event per replicate), we can then make an UpSet plot that shows how many bindings sites are supported by how many replicates.

```r
############################
# reproducibility of binding sites
############################
# import crosslink tracks per replicate in Rle format
sample1.minus.rle <- import.bw( bw_1_minus_path, as="Rle") %>% keepStandardChromosomes(pruning.mode = "c
sample2.minus.rle <- import.bw( bw_2_minus_path, as="Rle") %>% keepStandardChromosomes(pruning.mode = "c
sample3.minus.rle <- import.bw( bw_3_minus_path, as="Rle") %>% keepStandardChromosomes(pruning.mode = "c
sample4.minus.rle <- import.bw( bw_4_minus_path, as="Rle") %>% keepStandardChromosomes(pruning.mode = "c

sample1.plus.rle <- import.bw( bw_1_plus_path, as="Rle") %>% keepStandardChromosomes(pruning.mode = "coa
sample2.plus.rle <- import.bw( bw_2_plus_path, as="Rle") %>% keepStandardChromosomes(pruning.mode = "coa
sample3.plus.rle <- import.bw( bw_3_plus_path, as="Rle") %>% keepStandardChromosomes(pruning.mode = "coa
sample4.plus.rle <- import.bw( bw_4_plus_path, as="Rle") %>% keepStandardChromosomes(pruning.mode = "coa

# sum up the crosslink events per binding site for each replicate
# plus strand
bs.p = Binding_sites_n_cl_plus
bs.p$clp_rep1 = sample1.plus.rle[bs.p] %>% sum
bs.p$clp_rep2 = sample2.plus.rle[bs.p] %>% sum
bs.p$clp_rep3 = sample3.plus.rle[bs.p] %>% sum
bs.p$clp_rep4 = sample4.plus.rle[bs.p] %>% sum

# minus strand
bs.m = Binding_sites_n_cl_minus
bs.m$clp_rep1 = sample1.minus.rle[bs.m] %>% sum
bs.m$clp_rep2 = sample2.minus.rle[bs.m] %>% sum
bs.m$clp_rep3 = sample3.minus.rle[bs.m] %>% sum
bs.m$clp_rep4 = sample4.minus.rle[bs.m] %>% sum

# combine both strands
bs_cl_samples = c(bs.p, bs.m)

# give every binding an ID
bs_cl_samples$number <- 1:length(bs_cl_samples)

# make list of supported binding sites for the UpSet plot
UpSet_List = list(rep1 = bs_cl_samples[bs_cl_samples$clp_rep1>0]$number,
                  rep2 = bs_cl_samples[bs_cl_samples$clp_rep2>0]$number,
                  rep3 = bs_cl_samples[bs_cl_samples$clp_rep3>0]$number,
                  rep4 = bs_cl_samples[bs_cl_samples$clp_rep4>0]$number)

UpSetR::upset(UpSetR::fromList(UpSet_List), order.by = c("freq", "degree"), nsets = 4)
```

We can also compare the signal of the binding sites between two replicates in a scatter plot.

```r
# scatter plot for replicates 1 and 2
head(bs_cl_samples)
bs_cl_samples_df <- as.data.frame(bs_cl_samples)

# make plot
ggplot(bs_cl_samples_df, aes(x=clp_rep1, y=clp_rep2))+
  geom_point()
```

```r
# log scale
ggplot(bs_cl_samples_df, aes(x=log2(clp_rep1), y=log2(clp_rep2)))+
  geom_point()
```

```r
# point density
ggplot(bs_cl_samples_df, aes(x=log2(clp_rep1), y=log2(clp_rep2)))+
  geom_pointdensity()
```

```r
# calculate Pearson correlation
ggplot(bs_cl_samples_df, aes(x=log2(clp_rep1), y=log2(clp_rep2)))+
  geom_pointdensity()+
  stat_cor()
```

Exercise: Reproducibility of other replicates

Make a scatter plot for the other replicate combinations (replicate 1 against replicate 3, etc.).

1.1 Filter for reproducible binding sites

For our further analysis, we will focus only on binding sites reproducible in a certain number of replicates. Here, we decided to use binding sites that are supported by at least three out of four replicates:

```r
bs_cl_samples_df <- bs_cl_samples_df %>% rowwise %>%
  mutate(n_repro_reps = sum(clp_rep1>0,
                            clp_rep2>0,
                            clp_rep3>0,
                            clp_rep4>0))

head(bs_cl_samples_df)
```

```r
# how many binding sites are supported by less than 3 replicates?
nrow(bs_cl_samples_df[bs_cl_samples_df$n_repro_reps<3,])

# filter for 3 or more replicates
bs_cl_samples <- bs_cl_samples[bs_cl_samples_df$n_repro_reps>=3]

# use these as binding sites in the following analyses
Binding_sites <- bs_cl_samples
```

2 Combine binding sites with gene annotations to see where the RBP binds

2.1 Annotations

2.1.1 Load the corresponding gene annotation into R

We already used gene annotation to run STAR and PureCLIP as well as for visualization in IGV. To overlay our binding sites with the annotation, we should use the same version as above. The data sets of this project were all processed with the GENCODE gene annotation version 25 (hg38.p7). You can download this annotation from the GENCODE archive (ftp://ftp.ebi.ac.uk/pub/databases/gencode/Gencode_human/release_25/GRCh38.p7.genome.fa.gz). For this project we already loaded the gtf file (gtf) of this annotation to the GitHub data folder.

We will now load the annotation as a GRanges object:

```r
path_gtf <- "/path/to/your/folder/gencode.v25.annotation.gtf"
```

```r
gtf <- import(path_gtf)
gtf
```

2.1.2 Annotation overlap

If you look at the GRanges object with the annotations, you can see that the same region in the genome can overlap with multiple annotations. This has several reasons:

1. Each gene is annotated over multiple rows in the gtf file. The first row gives the coordinates and information for the complete ***gene***, followed by at least one more row for the ***transcript*** of this gene. For this transcript, there can be further rows specifying each ***exon*** and different types of ***gene region*** in the transcript eg., CDS or UTR. Each of these rows is represented as a separate row in the GRanges object. This means that most positions in a gene will have at least three annotations, ie. gene, transcript and region.
2. A gene can have more than one isoform due to alternative processing events, such as alternative splicing or alternative polyadenylation. Each of these transcript isoforms will have its own set of rows in the GRanges object. You can subset for all entries belonging to a gene by the ***gene_id*** column, and for all entries belonging to one transcript by the ***transcript_id*** column.

Exercise: How many annotations does you favorite gene have?

Take your favorite gene. How many transcript isoforms are annotated in the gtf file that we loaded? Are all of these isoforms protein-coding? What else do you find?

```r
# my favorite gene is PURA, the Ensembl gene_id is ENSG00000185129
# I can subset the GRanges objects by choosing all rows that have PURA in the gene_name column
anno_favorite_gene <- gtf[gtf$gene_name=="PURA"]
anno_favorite_gene
```

```
# with the table() function I can look at how many annotations of which type are there for PURA
table(anno_favorite_gene$type)

# GRanges offers a function that counts overlaps
countOverlaps(anno_favorite_gene)

# this tells us how many entries overlap with each row
# with the first row of anno_favorite_gene (this is the row of the gene) overlap 14 other entries (all)
# with the second row (one transcript) overlap 12 entries and so on
```

Exercise: How many genes overlap in the annotation? Are there genes that overlap? Let's see.

```
# select only genes from annotations
anno_genes <- gtf[gtf$type=="gene"]

# are ther overlaps?
head(countOverlaps(anno_genes), 20)

# yes many have overlaps eg. the gene in the 2. and the 3. row of the anno_genes granges
anno_genes[2:3]
```

In IGV we can see why:

!(Overlapping Genes)[/Users/melinaklostermann/Documents/iCLIP-course/Overlapping_genes.png]

2.2 Assign binding sites to the bound genes

How do we know on which of the overlapping genes the crosslink event really occurred? We don't.

There are several ways to prioritize among the overlapping genes. One possibility is to follow prior knowledge. For instance, if your RBP is known to bind to snoRNAs, you may favor a snoRNA gene over an overlapping protein-coding gene.

Alternatively, you can try to remove overlaps by filtering the annotation for the most reliable entries. For instance, we can use the level of the gene annotation provided by GENCODE in the column level: - 1: verified loci - 2: manually annotated loci - 3: automatically annotated loci A good way (in annotation for the human genome) is to go with genes of level 1 and 2 and to keep genes of level 3 only, when there is no overlap to a gene of gene level 1 or 2

Even after this filtering, there will still be occasional overlaps where two or more genes with level 1 and 2 coincide. In the following approach, we keep both. Alternatively, you could apply an additional filtering step, such as a hierarchy of annotations, to resolve these overlaps.

```
####################
# filter annotation
####################
#  first we filter the annotation for standard chromosomes
gtf <-keepStandardChromosomes(gtf, pruning.mode = "coarse")

# select genes from the annotation
gtf_genes <- gtf[gtf$type=="gene"]
# keep all genes with level 1 or 2
gtf_genes_GL12<- gtf_genes[gtf_genes$level <= 2]

# keep gene with level 3 only if they do not overlap
gtf_GL3 <- subsetByOverlaps(gtf_genes[gtf_genes$level==3], gtf_genes_GL12, type = "any", invert = T)
genes <- c(gtf_genes_GL12, gtf_GL3)
```

```r
######################
# which genes are hit
#####################
# subset genes with binding sites
hit_genes <- subsetByOverlaps(genes, Binding_sites)
hit_genes

# how many genes are hit?
NROW(hit_genes)

# make a data frame for ggplot that contains the gene_types
hit_genetypes_df <- data.frame(gene_types=hit_genes$gene_type)

# plot gene_types
ggplot(hit_genetypes_df , aes(x = gene_types)) +
  geom_bar(stat = "count") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) + # turns axis label 90 degree
  ggtitle("Gene target fractions") + # give plot a title
  xlab("labels") + # change label of x-axis
  ylab("counts [log10]") # and of y-axis
```

```r
# to see all bars, we can use a log scale on the y-axis

ggplot(hit_genetypes_df , aes(x = gene_types)) +
  geom_bar(stat = "count") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  ggtitle("Gene target fractions") +
  scale_y_log10() +
  xlab("labels") +
  ylab("counts [log10]")
```

2.3 Which regions of the genes are bound?

As a next step we can ask which regions within the transcripts are bound. We have already seen in IGV that some RBPs have a preference to bind for example at exon-intron boundaries or in the 3' or 5'UTR. These binding preferences are usually connected to the function of the RBP, and might help us discover the function of RBPs with unkown functionality.

However, most genes have several transcripts and we do not know which of them were bound. Therefore, a binding site can lie in one transcript in an intron and in an other transcript in the UTR and we will not know, which was the "true" bound region.

2.3.1 GenomicFeatures package for gene regions

You maybe saw already, that the annotation from the gtf file, does not distinguish 3' and 5'UTRs. Also it does not annoatate introns. Therefore, will will make use of another package called GenomicFeatures. It does not work on GRanges objects, but instead needs a txDb format. We can make a txDb file from our GRanges annotation with makeTxDbFromGranges:

```r
#make txDb
anno_txDb <- makeTxDbFromGRanges(gtf)
anno_txDb
```

With the txDb annotation we can now use GenomicFeatures functions to get the regions:

```
# get introns
introns <- intronsByTranscript(anno_txDb)
introns #this returns a GRangesList, but we want to have a GRanges again as output
introns <- unlist(introns) # this converts the GRnagesList in a GRanges again
introns

#3'UTR
utrs3 <- unlist(threeUTRsByTranscript(anno_txDb))

#5'UTR
utrs5 <- unlist(fiveUTRsByTranscript(anno_txDb))

#cds
cds <- cds(anno_txDb)
```

Lets see how often we get conflicting regions for a binding site.

```
#########################
### RBP exact location - Problem visualization
########################

### count the overlap of each binding site within each part of the gene
cds_BS =  countOverlaps(Binding_sites,cds)
introns_BS =  countOverlaps(Binding_sites, introns)
utrs3_BS =  countOverlaps(Binding_sites, utrs3)
utrs5_BS = countOverlaps(Binding_sites,utrs5)
count.df = data.frame(cds = cds_BS, intron = introns_BS, utr3 = utrs3_BS, utr5 = utrs5_BS)
head(count.df)

### plot the number of different transcript annotaitons at the same position
n_overlaps = apply(count.df, 1, function(x) length(x[x != 0])) # count how many regions overlap with ea
head(df)

# data.frame for ggplot
df <- data.frame(n_overlaps = n_overlaps)

#plot
ggplot(df, aes(x = n_overlaps)) +
  geom_bar(stat = "count") +
  ggtitle("Different transcript region overlaps") +
  xlab("number of different annotation") +
  ylab("counts")
```

2.3.2 Solution for mutliple regions overlapping the same binding site

There are several solutions to deal with these overlapping regions. One way would be to simply always choose the longest transcript of each gene or a random transcript of each gene. Another approach would be to see, what region a binding site has in most of the transcripts and then use this (magority vote). To resolve ties in this approach (eg. 2x utr and 2x intron), we can use a hierarchical rule based on our knowledge of the protein and our observations from IGV. FOR example, we could say for my RBP it is most likely that a binding site (with a tie of regions) is in the 3'UTR and least likely to be in the 5'UTR. And we could set up a rule like: 3'UTR > intron > cds > 5'UTR.

Exercise: Define a hierarchical rule for your RBP

What do you know about your RBP? Where did you see the highest peaks in IGV? Define which hierarchical rule you want to use.

```
###
### RBP exact location - Problem solution
###

### Setting the hierarchical rule for ties
rule = c("utr3", "intron", "cds", "utr5") # change this according to your rule

### Applying the majority vote
count.df.reg = count.df[, rule] %>% # this orders the columns in the order of our rule
 mutate(., outside = case_when(rowSums(.) == 0 ~ TRUE,
                               TRUE~FALSE))%>%
rowwise() %>%
  mutate(region = which.max(c(utr3,intron, cds, utr5))) %>%
  ungroup() %>%
  mutate( region = case_when( region == 1 ~ rule[1],
                              region == 2 ~ rule[2],
                              region == 3 ~ rule[3],
                              region == 4 ~ rule[4])
          )

head(count.df.reg)

Binding_sites$outside = count.df.reg$outside
Binding_sites$region = count.df.reg$region

Binding_sites_inside = Binding_sites[Binding_sites$outside==FALSE]

### Make plot
df = data.frame(region = Binding_sites_inside$region)
ggplot(df, aes(x = region)) +
  geom_bar() +
  xlab("region") +
  ylab("count")
```

Excercise for fast people: Impact of the hierarchical rule

Try out how different hierarchical rules influence your output. You can copy the code from above and set a new rule.

```
# save binding sites with annotation for tomorrow
path_annotated_BS <- "path/to/bindingsites/with/annotation.RData"
```

```
save(Binding_sites_inside, file=path_annotated_BS)
```