PlyRanges Test

2023-04-10

```r
# load libraries
library(tidyverse)
library(plyranges)
library(genomation)
library(biomaRt)
library(GenomicRanges)
library(kableExtra)
library(Hmisc)
library(varhandle)

# resolve namespace conflicts
library(conflicted)
conflicts_prefer(dplyr::filter)
conflicts_prefer(dplyr::select)
conflicts_prefer(dplyr::rename)
conflicts_prefer(base::intersect)
```

Introduction

This is a test of the plyranges controls for chromosome when doing joins. To do this, I will first determine whcat code can subset a granges object by chromosome. Then I wil run a for loop that finds the genes that have binary score equal to one on each chromosome. I will then compare this to the genes found when not looping by chromosome. If they are the same, I will conclude that plyranges does control for chromosome. If not, I will conclude the opposite.

```r
data_dir <- .get_config_path("LOCAL_SCEPTRE2_DATA_DIR")

# read in sceptre and seurat results
sceptre_path <- paste0(
  data_dir, "results/papalexi_analysis/",
  "sceptre_full_mrna_results_with_effect_size.rds"
)
seurat_path <- paste0(
  data_dir, "results/papalexi_analysis/",
  "seurat_all_perturbations_results.rds"
)
sceptre_results <- readRDS(sceptre_path)
seurat_results <- readRDS(seurat_path)

# read in K562 ChIP-seq data
stat1_filename <- "GSM1057011_STAT1peak_B.txt"
chipseq_fp <- paste0(data_dir, "data/chipseq/", stat1_filename)
```

```r
chipseq_data <- read.table(chipseq_fp)
colnames(chipseq_data) = c("chrom",'start_pos','end_pos',"pval","score",
                           "pos_max_peak","max_peak_height",
                           "rel_pos_max_peak_height","peak_size","mid_point",
                           "peak_to_mid_dist")

# read in genes from database
database_genes <- read_table(paste0(
  data_dir,
  "data/htftarget/dataset_3390.STAT1.target.txt"
)) |>
  suppressWarnings() |>
  pull(target_name)
```

```r
chipseq_data = GRanges(
  seqnames = chipseq_data$chrom,
  ranges = IRanges(start = chipseq_data$start_pos, end = chipseq_data$end_pos),
  score = chipseq_data$score,
  peak_position = chipseq_data$pos_max_peak)
```

```r
# get TSS for each gene
gene_names <- sceptre_results |> pull(response_id) |> unique() |> as.character()
ensembl <- useEnsembl(host = 'https://grch37.ensembl.org',
                      biomart = 'ENSEMBL_MART_ENSEMBL',
                      dataset = "hsapiens_gene_ensembl")
TSS_info <-getBM(attributes=c("external_gene_name", "chromosome_name", "start_position",
                       "end_position", "strand"),
                 filters=c('external_gene_name'),
                 value = gene_names, mart=ensembl) |>
  filter(chromosome_name %in% c(1:22, "X", "Y")) |>
  mutate(TSS = ifelse(strand == 1, start_position, end_position),
         chromosome_name = paste0("chr", chromosome_name)) |>
  rename(gene = external_gene_name, chr = chromosome_name) |>
  select(gene, chr, TSS)
```

```r
# get binary score for each gene
window_width_binary <- 5e3
TSS <- GRanges(
  seqnames = TSS_info$chr,
  ranges = IRanges(start = TSS_info$TSS-window_width_binary,
                   end = TSS_info$TSS+window_width_binary),
  gene = TSS_info$gene,
  TSS = TSS_info$TSS)

chipseq_scores_binary <-TSS |>
  join_overlap_left(chipseq_data) |>
  group_by(gene) |>
  summarise(binary_score = any(!is.na(score))) |>
  as_tibble()

chipseq_scores_binary = subset(chipseq_scores_binary,binary_score == T)
```

Finding Code That Filters By Chromosome

We now test to see if the filter function actually subsets each item by chromosome. We see that it does as the chromosome values only show chromosome one as we would hope

```
#get list of chromosomes
chr = sort(unique(varhandle::unfactor(TSS@seqnames@values)))
#try chromsome 1
test_chr = "chr1"
TSS_test = TSS %>%
  filter(seqnames == test_chr)
print(TSS_test@seqnames@values)
```

```
## [1] chr1
## 24 Levels: chr15 chr21 chr18 chr20 chr13 chr9 chrX chr22 chr8 chr7 ... chrY
```

Testing if Looping Through Genes Return the Same Genes as Not Looping

We now perform our loop. In our loop we filter the TSS and chipseq granges objects by chromosome. Then we perform the join operation and find which genes have a binary score equal to one. We compare this gene set to the gene set found without looping through each gene. Finally, we find the mean overlap between the two gene sets. If this overlap is equal to one, we know that both methods return the same gene sets.We do in fact see this and as such, conclude that plyranges does take into account chromosome

```
chr = sort(unique(varhandle::unfactor(TSS@seqnames@values)))
counter = 0
for(test_chr in chr){
  #get filtered TSS granges object
  TSS_filter = TSS %>%
  filter(seqnames == test_chr)
  #get filtered chipseq granges object
  chipseq_filter = chipseq_data%>%
    filter(seqnames == test_chr)
  #compute binary score
  chipseq_binary_temp <-TSS_filter |>
  join_overlap_left(chipseq_filter) |>
  group_by(gene) |>
  summarise(binary_score = any(!is.na(score))) |>
  as_tibble()
  #add to matrix
  if(counter == 0){
    chipseq_binary_test = chipseq_binary_temp
  }else{
    chipseq_binary_test = rbind(chipseq_binary_test,chipseq_binary_temp )
  }
  counter = counter + 1
}
#filter to only have genes whose score is 1
chipseq_binary_test = subset(chipseq_binary_test,binary_score == T)
#get gene lists
```

```r
filter_by_chromosome_genes = sort(as.matrix(chipseq_binary_test)[,1])
no_filter_by_chromosome_genes = sort(as.matrix(chipseq_scores_binary)[,1])
#see overlap. If overlap = 1, then we know that they return the same result
mean(filter_by_chromosome_genes==no_filter_by_chromosome_genes)
```

```
## [1] 1
```