Fluent genomics with plyranges and tximeta

Stuart Lee, Di Cook, Michael Lawrence, Michael Love

Abstract

In this workflow, we use the R/Bioconductor packages *tximeta* and *plyranges* for integrating results from an experiment using RNA-seq and ATAC-seq data. The *tximeta* package provides an automated way of importing RNA-seq transcript quantifications. It does this by creating a *SummarizedExperiment* object that knows the correct reference transcriptome and the metadata associated with the quantification software. Since the quantifications are transformed into a *SummarizedExperiment* object, they are ready for downstream analysis using many other Bioconductor packages. The *plyranges* package provides a framework for expressing operations on genomic coordinates such as finding overlaps and performing aggregations and genomic arithmetic. Here it is used to perform sensitivity analysis on the results of a differential expression and differential accessibility analysis with respect to their genomic context.

Introduction

To begin we will examine a subset of the RNA-seq and ATAC-seq data from Alasoo et al. [2018] - doi: 10.1038/s41588-018-0046-7. The experiment involved treatment of macrophage cell lines from a number of human donors with interferon (IFN) gamma, *Salmonella* infection, or both treatments combined. In the original study, the authors examined gene expression and chromatin accessibility in a subset of 86 successfully differentiated induced pluripotent stem cells (iPSC) lines, and examined baseline quantitative trait loci (QTL) and response QTL for both expression and accessibility. In the original study, it was found that there were expression QTLs that had an effect on chromatin accessibility in naive macrophage cells upon IFNg stimulation and consequently the variants implicated likely result in changes in binding of cell-specific transcription factors.

In this workflow, we will perform a much simpler analysis than the one found in Alasoo et al. [2018], using their publicly available RNA-seq and ATAC-seq data (ignoring the genotypes). We will examine the effect of IFNg stimulation on gene expression and chromatin accessibility, and look to see if there is an enrichment of differentially accessible (DA) ATAC-seq peaks in the vicinity of differentially expressed (DE) genes. This is plausible, as the transcriptomic response to IFNg stimulation may be mediated through binding of regulatory proteins to accessible regions, and this binding may increase the accessibility of those regions such that it can be detected by ATAC-seq.

Throughout the workflow, we will use existing Bioconductor infrastructure to understand these datasets. In particular, we will emphasize the use of the Bioconductor packages *plyranges* and *tximeta*. The first package is be used to perform easily-readable transformations of data tied to genomic ranges, e.g. shifts, windows, overlaps, etc. The *plyranges* package is described by Lee et al. [2019], and leverage underlying range operations described by Lawrence et al. [2013]. The second package described by Love et al. [2019] is used to read RNA-seq quantification into R/Bioconductor, such that the genomic ranges are automatically attached to the quantification data and differential expression results.

Experimental Data

The data used in this workflow is available from two packages: the *macrophage* Bioconductor ExperimentData package and from this workflow package *fluentGenomics* directly.

The *fluentGenomics* package contains functionality to download and generate a cached *SummarizedExperiment* object from the normalized ATAC-seq data provided by Alasoo and Gaffney [2017]. This object contains all 145 ATAC-seq samples across all experimental conditions as analyzed by Alasoo et al. [2018]. The data can be also be downloaded directly from the Zenodo deposition.

The following code loads the path to the cached data file, or if it is not present will create the cache and generate a *SummarizedExperiment* using the the *BiocFileCache* package [Shepherd and Morgan, 2019].

We can then read the cached file and assign it to an object called `atac`. Note that this step is not strictly necessary to run the workflow.

```r
library(fluentGenomics)
path_to_se <- cache_atac_se()
atac <- readRDS(path_to_se)
```

A precise description of how we obtained this *SummarizedExperiment* object can be found in section .

The *macrophage* package contains RNA-seq quantification from 24 RNA-seq samples, a subset of the RNA-seq samples generated and analyzed by Alasoo et al. [2018]. The paired-end reads were quantified using *Salmon* [Patro et al., 2017], using the Gencode 29 human reference transcripts [Frankish et al., 2018]. For more details on quantification, and the exact code used, consult the vignette of the macrophage package. The package also contains the `Snakemake` file that was used to distribute the *Salmon* quantification jobs on a cluster [Köster and Rahmann, 2012].

RNA-seq data analysis

Easy RNA-seq data import with *tximeta*

First, we specify a directory `dir`, where the quantification files are stored. You could simply specify this directory with:

```
dir <- "/path/to/quant/files"
```

where the path is relative to your current R session. However, here in this case we have distributed the files in the *macrophage* package. The relevant directory can be located and associated files can be located `system.file`.

```
dir <- system.file("extdata", package="macrophage")
```

Information about the experiment is contained in the `coldata.csv` file. We leverage the *dplyr* and *readr* packages (as part of the *tidyverse*) to read this file into R [Wickham et al., 2019]. We will see later that *plyranges* extends these packages to accommodate genomic ranges.

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(readr)
colfile <- file.path(dir, "coldata.csv")
coldata <- read_csv(colfile) %>%
  select(
    names,
    id = sample_id,
    line = line_id,
    condition = condition_name
  ) %>%
  mutate(
    files = file.path(dir, "quants", names, "quant.sf.gz"),
    condition = relevel(factor(condition), "naive")
  )
```

```
## Parsed with column specification:
## cols(
##   names = col_character(),
##   sample_id = col_character(),
##   line_id = col_character(),
##   replicate = col_double(),
##   condition_name = col_character(),
##   macrophage_harvest = col_character(),
##   salmonella_date = col_character(),
##   ng_ul_mean = col_double(),
##   rna_extraction = col_character(),
##   rna_submit = col_character(),
##   library_pool = col_character(),
##   chemistry = col_character(),
##   rna_auto = col_double()
## )
```

```
coldata
```

```
## # A tibble: 24 x 5
##    names      id     line  condition  files
##    <chr>      <chr>  <chr> <fct>      <chr>
##  1 SAMEA1038... diku_A diku... naive      /Library/Frameworks/R.framework/Versions/...
##  2 SAMEA1038... diku_B diku... IFNg       /Library/Frameworks/R.framework/Versions/...
##  3 SAMEA1038... diku_C diku... SL1344     /Library/Frameworks/R.framework/Versions/...
##  4 SAMEA1038... diku_D diku... IFNg_SL13... /Library/Frameworks/R.framework/Versions/...
##  5 SAMEA1038... eiwy_A eiwy... naive      /Library/Frameworks/R.framework/Versions/...
##  6 SAMEA1038... eiwy_B eiwy... IFNg       /Library/Frameworks/R.framework/Versions/...
##  7 SAMEA1038... eiwy_C eiwy... SL1344     /Library/Frameworks/R.framework/Versions/...
##  8 SAMEA1038... eiwy_D eiwy... IFNg_SL13... /Library/Frameworks/R.framework/Versions/...
##  9 SAMEA1038... fikt_A fikt... naive      /Library/Frameworks/R.framework/Versions/...
## 10 SAMEA1038... fikt_B fikt... IFNg       /Library/Frameworks/R.framework/Versions/...
## # ... with 14 more rows
```

After we have read the `coldata.csv` file, we select relevant columns from this table and create a new column called `files` and transform the existing `condition` column to a factor with the "naive" cell line as the baseline. The `files` column points to the quantifications for each observation - these files have been gzipped, but would typically not have the 'gz' ending if used from `salmon` directly. One other thing to note is the use of the pipe operator,`%>%`, which can be read as then i.e. first read the data, then select columns, then mutate them.

Now we have a data frame summarizing the experimental design and the locations of the quantifications, and are ready for import with *tximeta*. The following lines of code do a lot of work for the analyst: importing the RNA-seq quantification (dropping inferential replicates in this case), locating the relevant reference transcriptome, attaching the transcript ranges to the data, and fetching genome information. The result is stored in the Bioconductor data structure called a *SummarizedExperiment*.

```
suppressPackageStartupMessages(library(SummarizedExperiment))
library(tximeta)
se <- tximeta(coldata, dropInfReps=TRUE)
```

```
## importing quantifications

## reading in files with read_tsv

## 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
## found matching linked transcriptome:
## [ GENCODE - Homo sapiens - release 29 ]
## loading existing TxDb created: 2019-11-22 01:02:58
## Loading required package: GenomicFeatures
## Loading required package: AnnotationDbi
##
## Attaching package: 'AnnotationDbi'
##
## The following object is masked from 'package:dplyr':
##
##     select
##
## loading existing transcript ranges created: 2019-11-22 01:06:45
## fetching genome info for GENCODE
```

```
se
```

```
## class: RangedSummarizedExperiment
## dim: 205870 24
## metadata(6): tximetaInfo quantInfo ... txomeInfo txdbInfo
## assays(3): counts abundance length
## rownames(205870): ENST00000456328.2 ENST00000450305.2 ...
##   ENST00000387460.2 ENST00000387461.2
## rowData names(3): tx_id gene_id tx_name
## colnames(24): SAMEA103885102 SAMEA103885347 ... SAMEA103885308
##   SAMEA103884949
## colData names(4): names id line condition
```

The *tximeta* package can be used without an internet connection, in this case the linked transcriptome can be created directly from a salmon index and gtf.

```
makeLinkedTxome(
  indexDir=file.path(dir, "gencode.v29_salmon_0.12.0"),
  source="Gencode",
  organism="Homo sapiens",
  release="29",
  genome="GRCh38",
  fasta="ftp://ftp.ebi.ac.uk/pub/databases/gencode/Gencode_human/release_29/gencode.v29.transcripts.fa.gz",
  gtf=file.path(dir, "gencode.v29.annotation.gtf.gz"), # local version
  write=FALSE
)
```

Because *tximeta* knows the correct reference transcriptome, we can ask *tximeta* to summarize the transcript-level data to the gene level using the methods of Soneson et al. [2015].

```
gse <- summarizeToGene(se)
```

```
## loading existing TxDb created: 2019-11-22 01:02:58

## obtaining transcript-to-gene mapping from TxDb

## loading existing gene ranges created: 2019-11-23 02:30:13

## summarizing abundance

## summarizing counts

## summarizing length
```

Preliminary RNA-seq DE analysis

We can easily run a differential expression analysis with *DESeq2* using the following code chunks [Love et al., 2014]. The design indicates that we want to control for the donor (`line`) and test for differences in gene expression on the condition. For a more comprehensive discussion of DE analysis see Love et al. [2016] and Law et al. [2018].

```
library(DESeq2)
dds <- DESeqDataSet(gse, ~line + condition)
```

```
## using counts and average transcript lengths from tximeta

## Warning in DESeqDataSet(gse, ~line + condition): some variables in design
## formula are characters, converting to factors
```

```r
# filter out lowly expressed genes
# at least 10 counts in at least 6 samples
keep <- rowSums(counts(dds) >= 10) >= 6
dds <- dds[keep,]
```

Below we set the contrasts on the condition variable, indicating we are estimating log2 fold changes of IFNg stimulated cell lines against naive cell lines. We are interested in log fold changes greater than 1 at a false discovery rate at of 1%.

```r
dds <- DESeq(dds)
```

```
## estimating size factors

## using 'avgTxLength' from assays(dds), correcting for library size

## estimating dispersions

## gene-wise dispersion estimates

## mean-dispersion relationship

## final dispersion estimates

## fitting model and testing
```

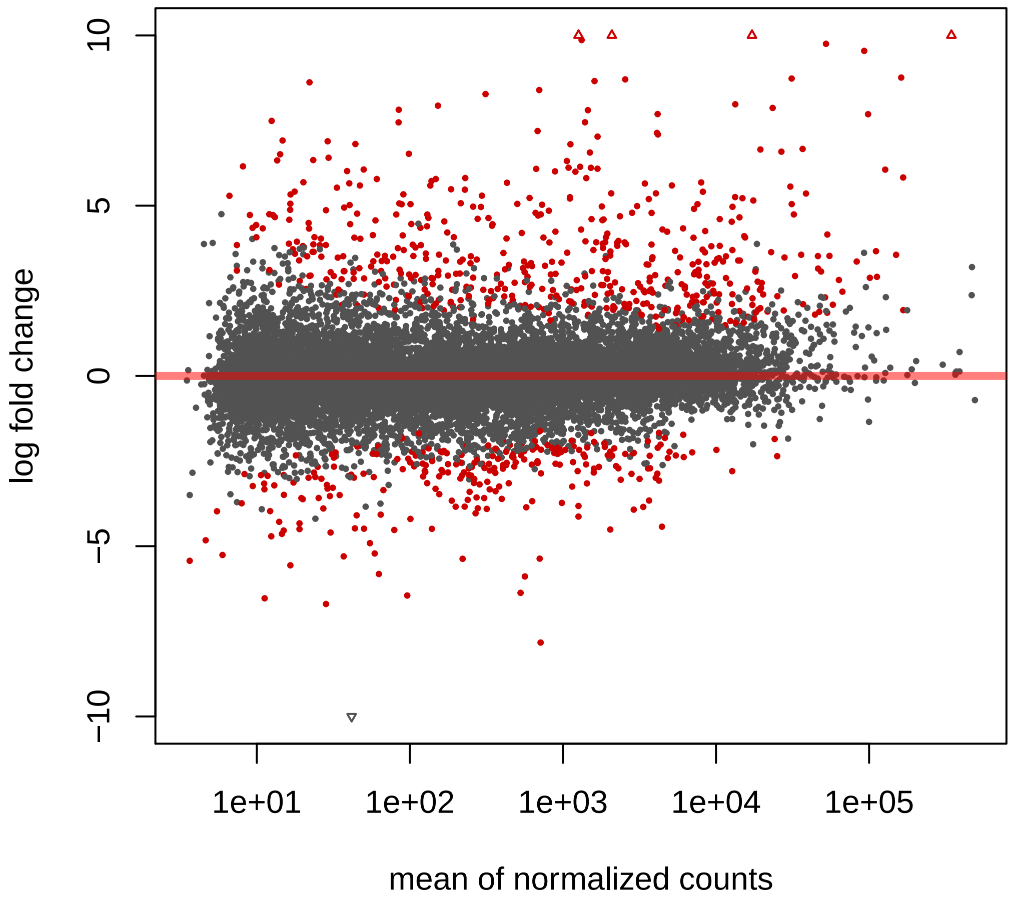
```r
res <- results(dds,
               contrast=c("condition","IFNg","naive"),
               lfcThreshold=1, alpha=0.01)
```

To see the results of the expression analysis, we can generate a summary table and an MA plot:

```r
summary(res)
```

```
##
## out of 17806 with nonzero total read count
## adjusted p-value < 0.01
## LFC > 1.00 (up)     : 502, 2.8%
## LFC < -1.00 (down) : 247, 1.4%
## outliers [1]        : 0, 0%
## low counts [2]      : 0, 0%
## (mean count < 3)
## [1] see 'cooksCutoff' argument of ?results
## [2] see 'independentFiltering' argument of ?results
```

```r
DESeq2::plotMA(res, ylim=c(-10,10))
```



We now output the results as a *GRanges* object, due to the conventions of *plyranges* we construct a new column called `gene_id` from the row names of the results. Each row now contains the genomic region (`seqnames`, `start`, `end`, `strand`) along with corresponding metadata columns (the `gene_id` and the results of the test). Note that *tximeta* has correctly identified the reference genome as "hg38", this has also been annotated onto the target *GRanges*. This kind of book-keeping is vital once overlap operations are performed to ensure that *plyranges*

```
suppressPackageStartupMessages(library(plyranges))
de_genes <- results(dds,
                    contrast=c("condition","IFNg","naive"),
                    lfcThreshold=1,
                    format="GRanges") %>%
  names_to_column("gene_id")
de_genes
```

```
## GRanges object with 17806 ranges and 7 metadata columns:
##           seqnames              ranges strand |            gene_id
##              <Rle>           <IRanges>  <Rle> |        <character>
##       [1]     chrX 100627109-100639991      - | ENSG00000000003.14
##       [2]    chr20   50934867-50958555      - | ENSG00000000419.12
##       [3]     chr1 169849631-169894267      - | ENSG00000000457.13
##       [4]     chr1 169662007-169854080      + | ENSG00000000460.16
##       [5]     chr1   27612064-27635277      - | ENSG00000000938.12
##       ...      ...                 ...    ... .                ...
##   [17802]    chr10   84167228-84172093      - |  ENSG00000285972.1
##   [17803]     chr6   63572012-63583587      + |  ENSG00000285976.1
##   [17804]    chr16   57177349-57181390      + |  ENSG00000285979.1
##   [17805]     chr8 103398658-103501895      - |  ENSG00000285982.1
##   [17806]    chr10   12563151-12567351      + |  ENSG00000285994.1
##                   baseMean     log2FoldChange             lfcSE
##                  <numeric>          <numeric>         <numeric>
##       [1] 171.570646163445 -0.282245015065582  0.300571026277417
##        [2] 967.751278980391 0.0391222756936352 0.0859707605047955
##        [3] 682.432885098654    1.2846178585311  0.196906721741941
##        [4] 262.963397841117  -1.47187616421189  0.218691645887265
##        [5] 2660.10225731917  0.675478091290521  0.236053041372838
##        ...              ...                ...                ...
##    [17802] 10.0474624496157  0.548451844773876  0.444318686394084
##    [17803] 4586.34616821518 -0.033929582570062  0.188004977365846
##    [17804] 14.2965310090402  0.312347650582085  0.522699844356108
##    [17805] 27.7629588245413  0.994518742790125   1.58237312176743
##    [17806] 6.60408582708505   0.25399752352481    0.5957511892896
##                        stat             pvalue               padj
##                   <numeric>          <numeric>          <numeric>
##        [1]                0                  1                  1
##        [2]                0                  1                  1
##        [3]  1.44544511235177  0.148332899695748                  1
##        [4] -2.15772377722715 0.0309493141635637 0.409727500369082
##        [5]                0                  1                  1
##        ...              ...                ...                ...
##    [17802]                0                  1                  1
##    [17803]                0                  1                  1
##    [17804]                0                  1                  1
##    [17805]                0                  1                  1
##    [17806]                0                  1                  1
##    -------
##    seqinfo: 25 sequences (1 circular) from hg38 genome
```

From this, we can restrict the results where that meet our FDR threshold and select (and rename) the metadata columns we're interested in:

```
de_genes <- de_genes %>%
  filter(padj < 0.01) %>%
  select(gene_id, de_log2FC = log2FoldChange, de_padj = padj)
```

Next we re-run `results` because we don't want to use an `lfcThreshold` this time. This will extract genes which are not differentially expressed according to the *DESeq2* significance test.

```
other_genes <- results(dds,
                       contrast=c("condition","IFNg","naive"),
                       format="GRanges") %>%
  filter(pvalue > 0.1) %>%
  names_to_column("gene_id") %>%
  dplyr::select(gene_id,
                   de_log2FC = log2FoldChange,
                   de_padj = padj)
```

ATAC-seq peak differential abundance analysis

The following section describes the process we have used for generating a *GRanges* object of differential peaks from that ATAC-seq data in Alasoo et al. [2018].

The code chunks for the remainder of this section are not run.

Generating a *SummarizedExperiment* object

The *SummarizedExperiment* object containing ATAC-seq peaks can be created from the following tab-delimited files found in [Alasoo and Gaffney, 2017]:

- The annotated peaks: `ATAC_peak_metadata.txt.gz` (5.6M)
- The sample metadata: `ATAC_sample_metadata.txt.gz` (<1M)
- The matrix of normalized read counts: `ATAC_cqn_matrix.txt.gz` (109M)

First, we read in the peak metadata (locations in the genome), and convert to a *GRanges* object. The `as_granges()` function automatically converts the data.frame into a *GRanges* object, from that result, we extract the peak_id column and set the genome information to the build "GRCh38". We know this from the Zenodo entry

```
peaks_df <- read_tsv("ATAC_peak_metadata.txt.gz",
  col_types = c("cidciicdc")
)

peaks_gr <- peaks_df %>%
  as_granges(seqnames = chr) %>%
  select(peak_id=gene_id) %>%
  set_genome_info(genome = "GRCh38")
```

We also read in the sample metadata:

```
atac_coldata <- read_tsv("ATAC_sample_metadata.txt.gz") %>%
 select(
    sample_id,
    donor,
    condition = condition_name
  ) %>%
  mutate(condition = relevel(factor(condition), "naive"))
```

The ATAC-seq data has already been normalized with *cqn* [Hansen et al., 2012] and log2 transformed. Loading the *cqn*-normalized matrix of log2 transformed read counts takes ~30 seconds and loads an object of ~370 Mb. We set the column names so the first row contains the rownames of the matrix, and the remaining columns are matched to the sample identifiers.

```
atac_mat <- read_tsv("ATAC_cqn_matrix.txt.gz",
                     skip = 1,
                     col_names =c("rownames", atac_coldata[["sample_id"]]))
rownames <- atac_mat[["rownames"]]
atac_mat <- as.matrix(atac_mat[,-1])
rownames(atac_mat) <- rownames
```

Finally, we then combine the data, and two pieces of metadata into a *SummarizedExperiment*:

```
atac <- SummarizedExperiment(list(cqndata=atac_mat),
                             rowRanges=peaks_gr,
                             colData=atac_coldata)
```

Using limma for differential accessibility

For assessing differential accessibility, we run *limma* [Smyth, 2004], and generate the a summary of log fold changes and adjusted p-values for the peaks:

```
library(limma)
design <- model.matrix(~donor + condition, colData(atac))
fit <- lmFit(assay(atac), design)
fit <- eBayes(fit)
idx <- which(colnames(fit$coefficients) == "conditionIFNg")
tt <- topTable(fit, coef=idx, sort.by="none", n=nrow(atac))
```

We now take the `rowRanges` of the *SummarizedExperiment* and attach the LFC and adjusted p-value from *limma*, so that we can consider the overlap with differential expression. Note that we set the genome build to "hg38" and restyle the chromosome information to use the "UCSC" style.

```
atac_peaks <- rowRanges(atac) %>%
  remove_names() %>%
  mutate(
    da_log2FC = tt$logFC,
    da_padj = tt$adj.P.Val
  ) %>%
  set_genome_info(genome = "hg38")

seqlevelsStyle(atac_peaks) <- "UCSC"
```

Finding overlaps with *plyranges*

We have already used *plyranges* a number of times above, to `filter`, `mutate` and `select` on *GRanges* objects, as well as ensuring the correct genome annotation and style has been used.

The final *GRanges* object containing the DA peaks is included in the workflow package and is loaded below:

```
library(fluentGenomics)
peaks
```

```
## GRanges object with 296220 ranges and 3 metadata columns:
##            seqnames              ranges strand |         peak_id
##               <Rle>           <IRanges>  <Rle> |     <character>
##        [1]     chr1          9979-10668      * |     ATAC_peak_1
##        [2]     chr1         10939-11473      * |     ATAC_peak_2
##        [3]     chr1         15505-15729      * |     ATAC_peak_3
##        [4]     chr1         21148-21481      * |     ATAC_peak_4
##        [5]     chr1         21864-22067      * |     ATAC_peak_5
##        ...      ...                 ...    ... .             ...
##   [296216]     chrX 155896572-155896835      * | ATAC_peak_296216
##   [296217]     chrX 155958507-155958646      * | ATAC_peak_296217
##   [296218]     chrX 156016760-156016975      * | ATAC_peak_296218
##   [296219]     chrX 156028551-156029422      * | ATAC_peak_296219
##   [296220]     chrX 156030135-156030785      * | ATAC_peak_296220
##                     da_log2FC              da_padj
##                     <numeric>            <numeric>
##        [1]  0.266185396736073 9.10672732956434e-05
##        [2]   0.32217712436691 2.03434717570469e-05
##        [3] -0.574159538548115 3.41707743345703e-08
##        [4]  -1.14706617895329 8.22298606986521e-26
##        [5] -0.896143162633654 4.79452571676397e-11
##        ...                ...                  ...
##   [296216] -0.834628897017445  1.3354605397165e-11
##   [296217] -0.147537281935847    0.313014754316915
##   [296218] -0.609732301631964 3.62338775135558e-09
##   [296219] -0.347678474957794 6.94823191242968e-06
##   [296220]  0.492442459200901 7.07663984067763e-13
##   -------
##   seqinfo: 23 sequences from hg38 genome; no seqlengths
```

For the overlap analysis, we filter these to have a nominal FDR bound of 1$.

```
da_peaks <- peaks %>%
  filter(da_padj < 0.01)
```

We now have *GRanges* objects that contain DE genes, genes without strong signal of DE, and DA peaks. We are ready to perform our original aim and answer the question: is there an enrichment of DA ATAC-seq peaks in the vicinity of DE genes?

Down sampling non-differentially expressed genes

As *plyranges* is built on top of *dplyr* it implements methods for many of it's verbs for *GRanges* objects. Here we can use, `slice` to randomly sample the rows of the `other_genes`. The `sample.int` function will generate random samples of size equal to the number of DE-genes from the number of rows in `other_genes`:

```r
size <- length(de_genes)
slice(other_genes, sample.int(n(), size))
```

```
## GRanges object with 749 ranges and 3 metadata columns:
##         seqnames              ranges strand |            gene_id
##            <Rle>           <IRanges>  <Rle> |        <character>
##     [1]     chr6   96521595-96555276      + |  ENSG00000014123.9
##     [2]    chr12   66188879-66254622      + | ENSG00000090376.10
##     [3]     chr3   44729596-44737083      + | ENSG00000186446.11
##     [4]     chr7 157138913-157269372      + | ENSG00000009335.17
##     [5]     chr7     6577434-6589374      + | ENSG00000136247.14
##     ...      ...                 ...    ... .                ...
##   [745]    chr16   30378106-30400108      + | ENSG00000180035.12
##   [746]    chr19   19865886-19894674      + |  ENSG00000256771.3
##   [747]     chrX 119871487-119876662      + |  ENSG00000125356.6
##   [748]     chr2 113705011-113756823      - | ENSG00000115084.13
##   [749]    chr11 119206276-119313926      + |  ENSG00000110395.6
##                   de_log2FC           de_padj
##                   <numeric>         <numeric>
##     [1]   0.157288238952329 0.393474621468054
##     [2]   0.192675255367846 0.638575038094171
##     [3]   0.294974004663957 0.677850298147425
##     [4]  0.0541851010739642 0.688618637431238
##     [5]  -0.168092883269452 0.412214659609548
##     ...                 ...               ...
##   [745]  -0.253169808554397 0.437374379292198
##   [746]  -0.288346936786858 0.321576385838034
##   [747] -0.0656250155572518 0.660307119203159
##   [748]  0.0920937558644995  0.78240487118216
##   [749]   0.111941856478517 0.649460473096424
##   -------
##   seqinfo: 25 sequences (1 circular) from hg38 genome
```

We can repeat this many times to create many samples via replicate:

```r
# set a seed for the results
set.seed(2019-08-02)
boot_genes <- replicate(10,
                        slice(other_genes, sample.int(n(), size)),
                        simplify = FALSE)
```

This creates a list of *GRanges* objects as a list, we can bind these together using the `bind_ranges` function. This function creates a new column called "resample" on the result that identifies each of the input *GRanges* objects:

```r
boot_genes <- bind_ranges(boot_genes, .id = "resample")
```

Similarly, we can then combine the `boot_genes` *GRanges*, with the DE *GRanges* object. As the resample column was not present on the DE *GRanges* object, this is given a missing value which we recode to a 0 using `mutate()`

```r
all_genes <- bind_ranges(
  de=de_genes,
  not_de = boot_genes,
  .id="origin"
  ) %>%
  mutate(
    origin = factor(origin, c("de", "not_de")),
    resample = ifelse(is.na(resample), 0L, as.integer(resample))
  )
all_genes
```

```
## GRanges object with 8239 ranges and 5 metadata columns:
##           seqnames              ranges strand |            gene_id
##              <Rle>            <IRanges>  <Rle> |        <character>
##      [1]     chr1 196651878-196747504      + | ENSG00000000971.15
##      [2]     chr6   46129993-46146699      + |  ENSG00000001561.6
##      [3]     chr4   17577192-17607972      + | ENSG00000002549.12
##      [4]     chr7 150800403-150805120      + |  ENSG00000002933.8
##      [5]     chr4   15778275-15853230      + | ENSG00000004468.12
##      ...      ...                 ...    ... .                ...
##   [8235]     chr3   72749277-72861914      - | ENSG00000144736.13
##   [8236]    chr17   29566052-29573157      + | ENSG00000167543.15
##   [8237]     chr7 129225023-129430211      + | ENSG00000158467.16
##   [8238]     chr2   24029340-24049575      - | ENSG00000163026.11
##   [8239]    chr16     1826941-1840207      + | ENSG00000180185.11
##                   de_log2FC             de_padj  resample   origin
##                   <numeric>           <numeric> <integer> <factor>
##      [1]   4.98711071930695 1.37057050625117e-13         0       de
##      [2]   1.92721595378787  3.1747750217733e-05         0       de
##      [3]   2.93372501059128  2.0131038573066e-11         0       de
##      [4]   3.16721751137972 1.07359906028984e-08         0       de
##      [5]   5.40894352968188 4.82904694023763e-18         0       de
##      ...                ...                  ...       ...      ...
##   [8235] -0.324057975853188    0.531385106367204        10   not_de
##   [8236] 0.0582048743660616    0.681056515870893        10   not_de
##   [8237]  0.284556421479042    0.378275048382877        10   not_de
##   [8238] -0.130576070173704    0.601621121971093        10   not_de
##   [8239]  0.173903843804872    0.568644093307946        10   not_de
##   -------
##   seqinfo: 25 sequences (1 circular) from hg38 genome
```

Expanding genomic coordinates around the transcription start site

Now we would like to modify our gene ranges so their width is 10 kilobases on either side of their transcription start site (TSS). There are many ways one could do this but we prefer an approach via the anchoring methods in *plyranges*. Because there is a mutual dependence between the start, end, width and strand of a *GRanges* object, we define anchors to fix one of start and end, while modifying the width. As an example to extract just the TSS, we can anchor by the 5' end of the range and modify the width of the range to equal 1.

```r
all_genes <- all_genes %>%
  anchor_5p() %>%
  mutate(width = 1)
```

Anchoring by the 5' end of a range will fix the end of negatively stranded ranges, and fix the start of positively stranded ranges.

We can then repeat the same pattern but this time using `anchor_center()` to tell *plyranges* that we are making the TSS the midpoint of a range that has total width of 20kb or 10kb both upstream and downstream of the TSS.

```r
all_genes <-all_genes %>%
  anchor_center() %>%
  mutate(width=2*1e4)
```

Use overlap joins to find relative enrichment

We are now ready to compute overlaps between RNA-seq genes (our DE set and bootstrap samples) and the ATAC-seq peaks. In *plyranges*, overlaps are defined as joins between two *GRanges* objects: a *left* and a *right* *GRanges* object. In an overlap join, a match is any range on the *left GRanges* that is overlapped by the *right GRanges*. One powerful aspect of the overlap joins is that the result maintains all (metadata) columns from each of the *left* and *right* ranges which makes downstream summaries easy to compute.

To combine the DE genes with the DA peaks, we perform a left overlap join. This returns to us the `all_genes` ranges (potentially with duplication), but with the metadata columns from those overlapping DA peaks. For any gene that has no overlaps, the DA peak columns will have `NA`'s.

```r
overlap_genes <- all_genes %>%
  join_overlap_left(da_peaks)
overlap_genes
```

```
## GRanges object with 27591 ranges and 8 metadata columns:
##           seqnames              ranges strand |           gene_id
##              <Rle>           <IRanges>  <Rle> |       <character>
##       [1]     chr1 196641878-196661877      + | ENSG00000000971.15
##       [2]     chr6   46119993-46139992      + |  ENSG00000001561.6
##       [3]     chr4   17567192-17587191      + | ENSG00000002549.12
##       [4]     chr4   17567192-17587191      + | ENSG00000002549.12
##       [5]     chr4   17567192-17587191      + | ENSG00000002549.12
##       ...      ...                 ...    ... .                ...
##   [27587]    chr17   29556052-29576051      + | ENSG00000167543.15
##   [27588]     chr7 129215023-129235022      + | ENSG00000158467.16
##   [27589]     chr2   24039575-24059574      - | ENSG00000163026.11
##   [27590]    chr16     1816941-1836940      + | ENSG00000180185.11
##   [27591]    chr16     1816941-1836940      + | ENSG00000180185.11
##                   de_log2FC             de_padj  resample   origin
##                   <numeric>           <numeric> <integer> <factor>
##       [1]   4.98711071930695 1.37057050625117e-13         0       de
##       [2]   1.92721595378787  3.1747750217733e-05         0       de
##       [3]   2.93372501059128  2.0131038573066e-11         0       de
##       [4]   2.93372501059128  2.0131038573066e-11         0       de
##       [5]   2.93372501059128  2.0131038573066e-11         0       de
##       ...                ...                  ...       ...      ...
##   [27587] 0.0582048743660616    0.681056515870893        10   not_de
##   [27588]  0.284556421479042    0.378275048382877        10   not_de
##   [27589] -0.130576070173704    0.601621121971093        10   not_de
##   [27590]  0.173903843804872    0.568644093307946        10   not_de
##   [27591]  0.173903843804872    0.568644093307946        10   not_de
##                   peak_id          da_log2FC              da_padj
##               <character>          <numeric>            <numeric>
##       [1]  ATAC_peak_21236 -0.546582189082724 0.000115273676444232
##       [2] ATAC_peak_231183   1.45329684862127   9.7322474682763e-17
##       [3] ATAC_peak_193578  0.222371496904895 3.00939005719989e-11
##       [4] ATAC_peak_193579 -0.281615137872819 7.99888515457195e-05
##       [5] ATAC_peak_193580  0.673705317951604 7.60042918890061e-15
##       ...              ...                ...                  ...
##   [27587] ATAC_peak_109304  0.211750928531232  0.00111289505290053
##   [27588] ATAC_peak_255700  0.177364068655037 5.25384772617076e-09
##   [27589] ATAC_peak_133247 -0.266265981992122  1.0155043266685e-07
##   [27590]  ATAC_peak_97065 -0.405271356218346 1.28054921946532e-05
##   [27591]  ATAC_peak_97067  0.289105317951603 5.87369828076092e-07
##   -------
##   seqinfo: 25 sequences (1 circular) from hg38 genome
```

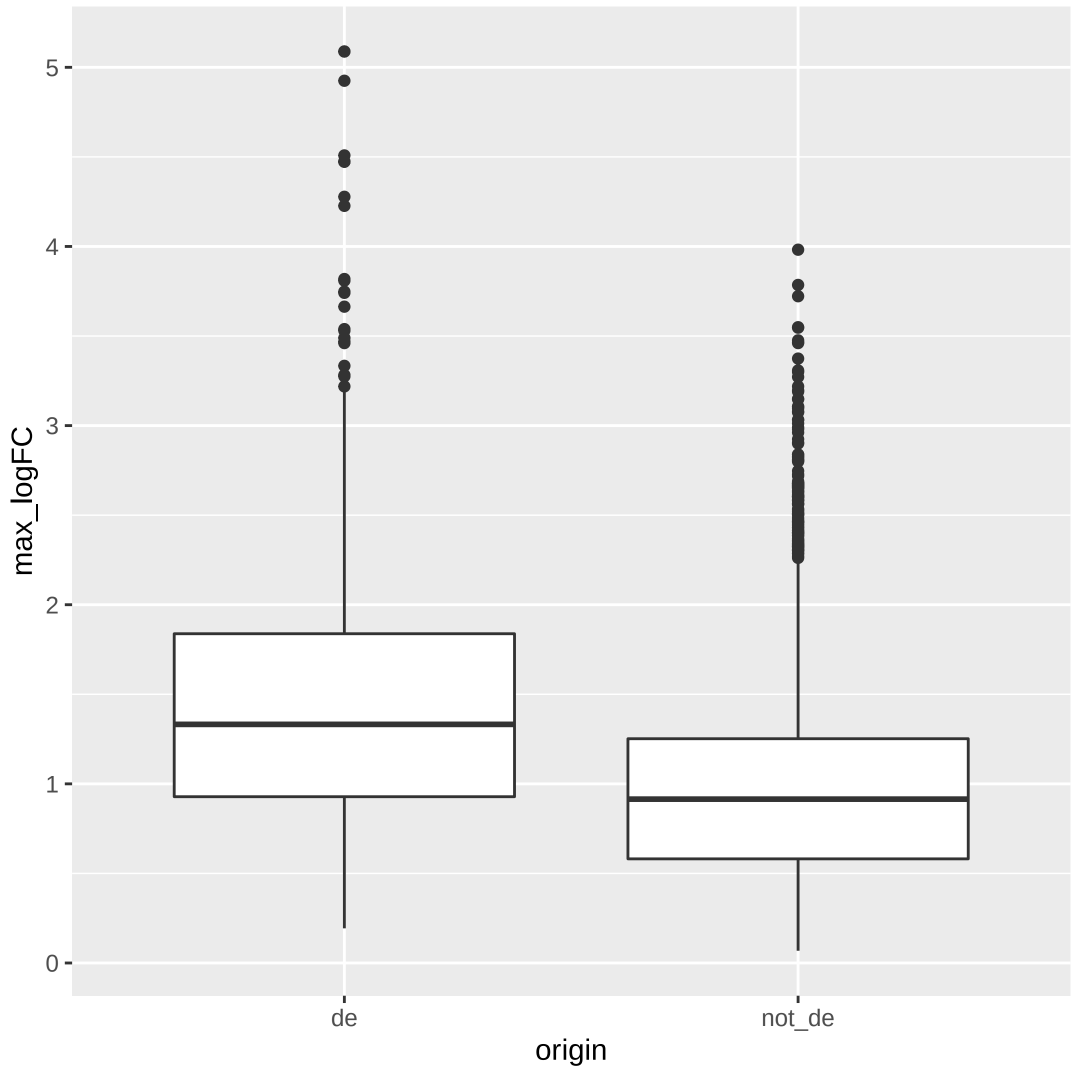
Now we can ask, how many DA peaks are near DE genes relative to "other" non-DE genes? A gene may appear more than once, since multiple peaks may overlap a single gene or because we have re-sampled the same gene more than once.

For each gene (that is the combination of chromosome, the start, end and strand), and the "origin" (DE vs not-DE) we can compute the distinct number of peaks for each gene and the maximum peak based on log FC. This is achieved via `reduce_ranges_directed`, which allows an aggregation to result in a GRanges object via merging neighboring genomic regions. The use of the directed suffix indicates we're maintaining strand information. In this case, we are simply merging ranges via the groups we mentioned above. We also have to account for the number of resamples we have performed when counting if there are any peaks, to ensure we do not double count the same peak:

```r
any_peaks <- overlap_genes %>%
  group_by(gene_id, origin)  %>%
  reduce_ranges_directed(
    any = sum(!is.na(da_padj)) / n_distinct(resample),
    max_logFC = max(abs(da_log2FC))
  )
```

We can then filter genes if there have any peaks and compare the peak fold changes between non-DE and DE genes using a boxplot:

```
library(ggplot2)
any_peaks %>%
  filter(any > 0) %>%
  as.data.frame() %>%
  ggplot(aes(origin, max_logFC)) +
  geom_boxplot()
```



In general, the DE genes have larger DA fold changes relative to the non-DE genes.

Next we examine how changes in DA LFC alter enrichment for DE genes. First, we want to know how the number of peaks within DE genes and non-DE genes change as we change threshold values on the peak LFC. As an example, we could compute this by arbitrarily chosen LFC thresholds as follows:

```
overlap_tab <- overlap_genes %>%
  group_by(origin) %>%
  summarize(any=sum(!is.na(da_padj)) / n_distinct(resample),
            lfc1 =sum(abs(da_log2FC) > 1, na.rm=TRUE)/ n_distinct(resample),
            lfc2= sum(abs(da_log2FC) > 2, na.rm=TRUE)/ n_distinct(resample))
overlap_tab
```

```
## DataFrame with 2 rows and 4 columns
##     origin       any      lfc1      lfc2
##   <factor> <numeric> <numeric> <numeric>
## 1       de      3416      1097       234
## 2   not_de    2362.3     443.8      32.3
```

Then using `summarize_all()` from *dplyr* we divide the rows of the above computation, to see that the relative enrichment increases for a 'larger' LFC:

```
overlap_tab %>%
  as.data.frame() %>%
  select(-origin) %>%
  summarize_all(.funs = ~Reduce("/", .))
```

```
##        any     lfc1     lfc2
## 1 1.446048 2.471834 7.244582
```

Due to the one-to-many mappings of DE genes to peaks, it is unknown if we have the same number of DE genes participating or less, as we increase the threshold on the peak LFC. This can be accounted for by grouping and aggregating twice. First, the number of peaks that meet the thresholds are computed within each gene, origin and resample group. Second, within the origin column, we compute the total number of peaks that meet the target threshold and the number of genes that have more than zero peaks (again averaging over the number of resamples).

```
overlap_genes %>%
  group_by(gene_id, origin, resample) %>%
  reduce_ranges_directed(
    lfc1 =sum(abs(da_log2FC) > 1, na.rm=TRUE),
    lfc2= sum(abs(da_log2FC) > 2, na.rm=TRUE)
  ) %>%
  group_by(origin) %>%
  summarize(
    lfc1_gene_count = sum(lfc1 > 0) / n_distinct(resample),
    lfc1_peak_count = sum(lfc1) / n_distinct(resample),
    lfc2_gene_count = sum(lfc2 > 0) / n_distinct(resample),
    lfc2_peak_count = sum(lfc2) / n_distinct(resample)
  )
```

```
## DataFrame with 2 rows and 5 columns
##     origin lfc1_gene_count lfc1_peak_count lfc2_gene_count lfc2_peak_count
##   <factor>       <numeric>       <numeric>       <numeric>       <numeric>
## 1       de             515            1097             151             234
## 2   not_de           301.4           443.8            30.4            32.3
```

To do this for many thresholds is cumbersome and would create a lot of duplicate code, instead we create a single function called `count_above_threshold` that accepts a GRanges object, and computes the peak count that meets the threshold.

```
count_above_threshold <- function(.data, threshold) {
  reduce_ranges_directed(.data,
                         threshold = threshold,
                         value = sum(abs(da_log2FC) > threshold, na.rm = TRUE)
  )
}
```

The above function will compute the counts for any arbitrary threshold, now we need to apply it over possible LFC thresholds of interest. We choose a grid of one hundred thresholds based on the range of absolute LFC values in the `da_peaks` *GRanges* object:

```
thresholds <- da_peaks %>%
  mutate(abs_lfc = abs(da_log2FC)) %>%
  with(
    seq(min(abs_lfc), max(abs_lfc), length.out = 100)
  )
```

The peaks are computed for each value by applying `count_above_threshold()` on the grouped *GRanges* object for each threshold and then binding the result. This could be done easily in parallel with `BiocParallel` but here we compute everything serially.

```
by_gene_origin <- overlap_genes %>%
  group_by(gene_id, origin, resample)

all_thresholds <- bind_ranges(
  lapply(thresholds, count_above_threshold, .data = by_gene_origin)
)
```

This creates a very *long* GRanges object. To compute the peak and gene counts for each threshold, we apply the same summarization as before:

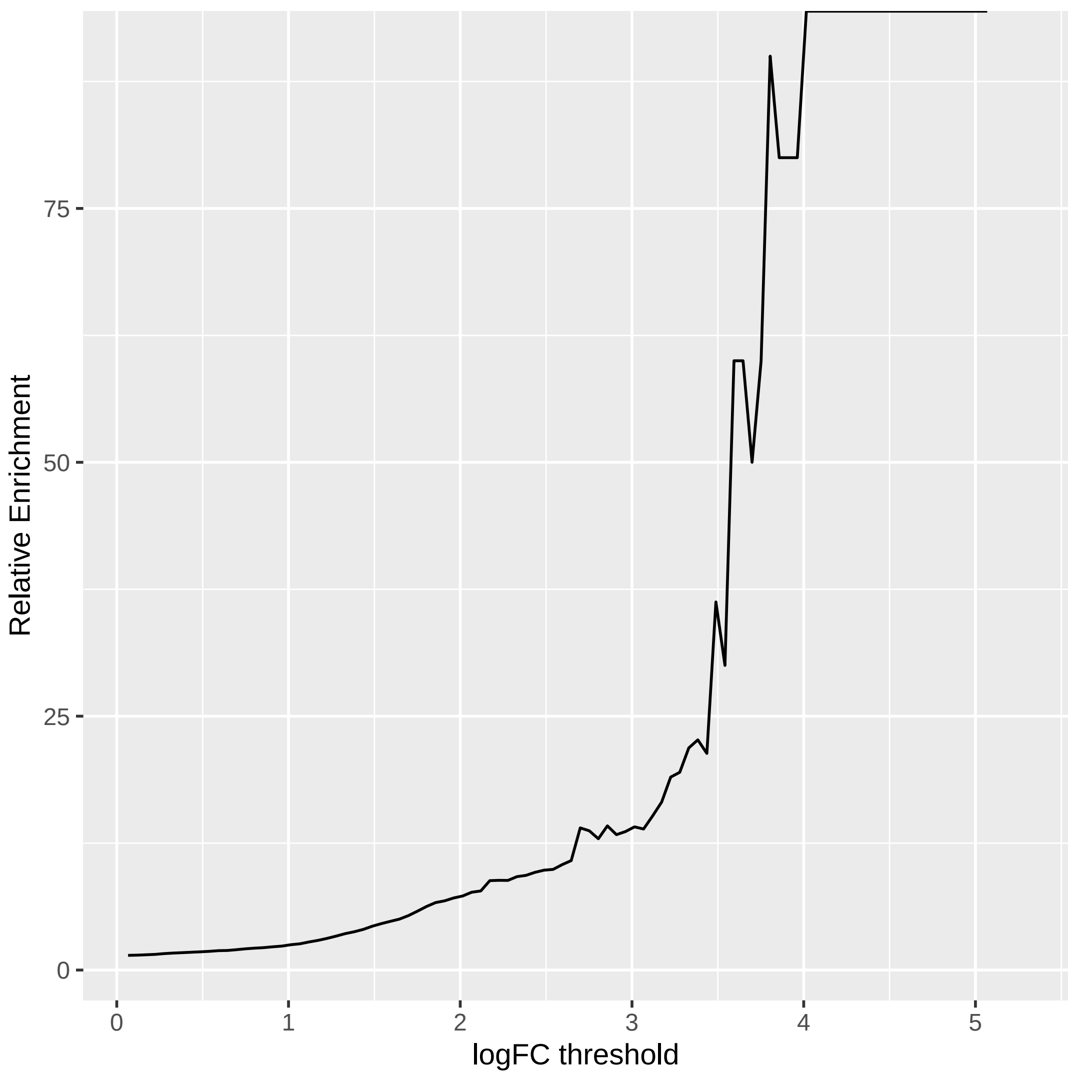
```
counts_by_threshold <- all_thresholds %>%
  group_by(origin, threshold) %>%
  summarize(
    gene_count = sum(value > 0) / n_distinct(resample),
    peak_count = sum(value) / n_distinct(resample)
  )
counts_by_threshold
```

```
## DataFrame with 200 rows and 4 columns
##       origin          threshold gene_count peak_count
##     <factor>          <numeric>  <numeric>  <numeric>
## 1         de 0.0658243106359027        727       3416
## 2         de  0.118483961449043        727       3392
## 3         de  0.171143612262182        727       3304
## 4         de  0.223803263075322        724       3160
## 5         de  0.276462913888462        723       3045
## ...      ...                ...        ...        ...
## 196   not_de   5.06849113788419          0          0
## 197   not_de   5.12115078869733          0          0
## 198   not_de   5.17381043951047          0          0
## 199   not_de   5.22647009032361          0          0
## 200   not_de   5.27912974113675          0          0
```

Again we can compute the relative enrichment in LFCs in the same manner as before and visualize how enrichment changes as the threshold value increases:

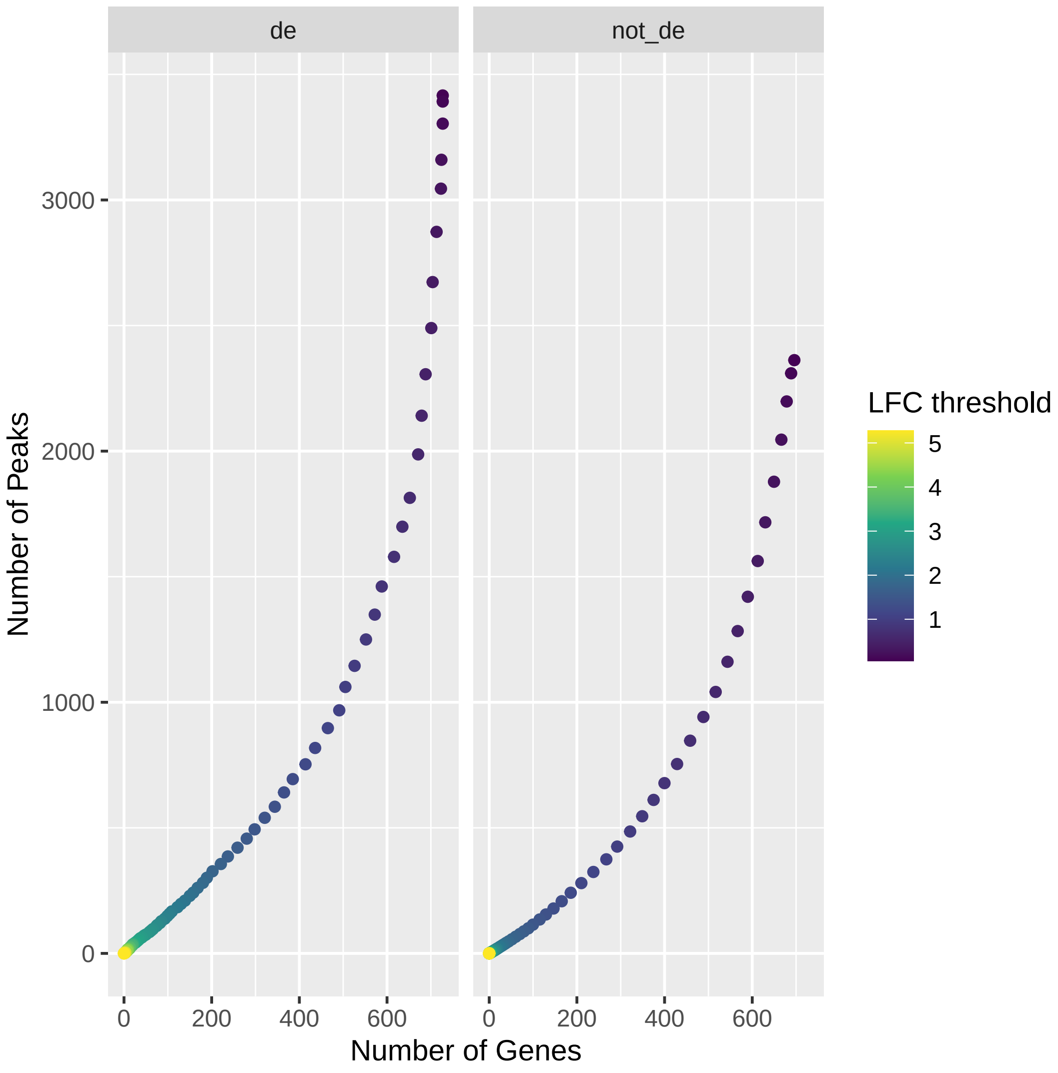
```
enrichment <- counts_by_threshold %>%
  as.data.frame() %>%
  group_by(threshold) %>%
  summarize(enrichment = Reduce("/", peak_count))

enrichment %>%
  ggplot(aes(x = threshold, y = enrichment)) +
  geom_line() +
  labs(x = "logFC threshold", y = "Relative Enrichment")
```

```
## Warning: Removed 4 rows containing missing values (geom_path).
```



We computed the sum of DA peaks near the DE genes, for increasing LFC thresholds on the accessibility change. As we increased the threshold, the number of total peaks went down (likewise the mean number of DA peaks per gene). It is also likely the number of DE genes with a DA peak nearby with such a large change went down - we can check this by plotting the number of DE genes against the number of DA peaks:

```r
counts_by_threshold %>%
  as.data.frame() %>%
  ggplot(aes(x = gene_count,
             y = peak_count,
             color = threshold)) +
  geom_point() +
  scale_color_viridis_c() +
  facet_wrap(~ origin) +
  labs(x = "Number of Genes",
       y = "Number of Peaks",
       color = "LFC threshold")
```



Discussion

We have shown that using *plyranges* and *tximeta* (with support of Bioconductor and *tidyverse* packages) that we can fluently iterate through the biological data science workflow: from import, through to modeling, wrangling and visualization.

Using *tximeta*, we have shown that it is straightforward to import RNA-seq quantification data, and that by ensuring the proper metadata is associated with it, we can guard against any mistakes in downstream analyses.

Using *plyranges*, we have extended the principles of the *tidyverse* to genomic ranges, and that by design we can leverage those packages to understand data measured along the genome. We have shown that analyses performed with *plyranges* clearly and (relatively) concisely express their intent; in most cases the code we have written closely matches it's description in English and clarifies how the features of a genomic range is being modified.

There are several further steps that would be interesting to perform in this analysis; for example, we could modify window size around the TSS to see how it effects enrichment and vary the cut-offs applied to FDR percentages applied to both the DE and DA peaks.

Software Availability

The workflow materials, including this article can be fully reproduced following the instructions found at the Github repository sa-lee/fluentGenomics. Moreover, the workflow and all downstream dependencies can be installed

This article and the analyses were performed with R [R Core Team, 2019] using the *rmarkdown* [Allaire et al., 2019], *knitr* [Xie, 2019, 2015] and *BiocWorkflowTools* [Smith et al., 2018] packages.

Session Info

```r
sessionInfo()
```

```
## R version 3.6.1 (2019-07-05)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS Mojave 10.14.6
##
## Matrix products: default
## BLAS:   /System/Library/Frameworks/Accelerate.framework/Versions/A/Frameworks/vecLib.framework/Versions/A/li
## LAPACK: /Library/Frameworks/R.framework/Versions/3.6/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_AU.UTF-8/en_AU.UTF-8/en_AU.UTF-8/C/en_AU.UTF-8/en_AU.UTF-8
##
## attached base packages:
## [1] parallel  stats4    stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] ggplot2_3.2.1.9000          plyranges_1.7.6
##  [3] DESeq2_1.26.0               GenomicFeatures_1.38.0
##  [5] AnnotationDbi_1.48.0        SummarizedExperiment_1.16.0
##  [7] DelayedArray_0.12.0         BiocParallel_1.20.0
##  [9] matrixStats_0.55.0          Biobase_2.46.0
## [11] GenomicRanges_1.38.0        GenomeInfoDb_1.22.0
## [13] IRanges_2.20.1              S4Vectors_0.24.1
## [15] BiocGenerics_0.32.0         readr_1.3.1
## [17] dplyr_0.8.3                 tximeta_1.4.2
## [19] fluentGenomics_0.0.3
##
## loaded via a namespace (and not attached):
##   [1] colorspace_1.4-1        htmlTable_1.13.3         XVector_0.26.0
##   [4] base64enc_0.1-3         fs_1.3.1                 rstudioapi_0.10
##   [7] farver_2.0.1            bit64_0.9-7              fansi_0.4.0
##  [10] xml2_1.2.2              splines_3.6.1            tximport_1.14.0
##  [13] geneplotter_1.64.0      knitr_1.26               zeallot_0.1.0
##  [16] Formula_1.2-3           jsonlite_1.6             Rsamtools_2.2.1
##  [19] annotate_1.64.0         cluster_2.1.0            dbplyr_1.4.2
##  [22] compiler_3.6.1          httr_1.4.1               backports_1.1.5
##  [25] assertthat_0.2.1        Matrix_1.2-18            lazyeval_0.2.2
##  [28] cli_2.0.0               acepack_1.4.1            htmltools_0.4.0
##  [31] prettyunits_1.0.2       tools_3.6.1              gtable_0.3.0
##  [34] glue_1.3.1              GenomeInfoDbData_1.2.2   rappdirs_0.3.1
##  [37] BiocWorkflowTools_1.12.0 Rcpp_1.0.3              vctrs_0.2.0
##  [40] Biostrings_2.54.0       rtracklayer_1.46.0       xfun_0.11
##  [43] stringr_1.4.0           lifecycle_0.1.0          ensembldb_2.10.2
##  [46] XML_3.98-1.20           zlibbioc_1.32.0          scales_1.1.0
##  [49] hms_0.5.2               ProtGenerics_1.18.0      AnnotationFilter_1.10.0
##  [52] RColorBrewer_1.1-2      yaml_2.2.0               curl_4.3
##  [55] memoise_1.1.0           gridExtra_2.3            biomaRt_2.42.0
##  [58] rpart_4.1-15            hunspell_3.0             latticeExtra_0.6-28
##  [61] stringi_1.4.3           RSQLite_2.1.4            genefilter_1.68.0
##  [64] checkmate_1.9.4         rlang_0.4.2              pkgconfig_2.0.3
##  [67] commonmark_1.7          bitops_1.0-6             evaluate_0.14
##  [70] lattice_0.20-38         purrr_0.3.3              labeling_0.3
##  [73] GenomicAlignments_1.22.1 htmlwidgets_1.5.1       bit_1.1-14
##  [76] tidyselect_0.2.5        magrittr_1.5             bookdown_0.16
##  [79] R6_2.4.1                spelling_2.1             Hmisc_4.3-0
##  [82] DBI_1.0.0               withr_2.1.2              pillar_1.4.2
##  [85] foreign_0.8-72          survival_3.1-8           RCurl_1.95-4.12
##  [88] nnet_7.3-12             tibble_2.1.3             crayon_1.3.4
##  [91] utf8_1.1.4              BiocFileCache_1.10.2     rmarkdown_1.18
##  [94] progress_1.2.2          usethis_1.5.1           locfit_1.5-9.1
##  [97] grid_3.6.1              data.table_1.12.8       blob_1.2.0
## [100] git2r_0.26.1            digest_0.6.23           xtable_1.8-4
## [103] openssl_1.4.1           munsell_0.5.0           viridisLite_0.3.0
## [106] askpass_1.1
```

Author Contributions

SL, DC, MIL and ML wrote the workflow.

Competing interests

The authors declare that they have no competing interests.

Funding

SL is supported by an Australian Government Research Training Program (RTP) scholarship with a top up scholarship from CSL Limited.

I confirm that the funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Acknowledgements

The authors would like to thank all participants of the Bioconductor 2019 and BiocAsia 2019 conferences who attended and provided feedback on early versions of this workflow paper.

References

K Alasoo, J Rodrigues, S Mukhopadhyay, AJ Knights, AL Mann, K Kundu, HIPSCI-Consortium, C Hale, Dougan G, and DJ Gaffney. Shared genetic effects on chromatin and gene expression indicate a role for enhancer priming in immune response. *Nature Genetics*, 50:424–431, 2018. doi: 10.1038/s41588-018-0046-7.

Kaur Alasoo and Daniel Gaffney. Processed read counts from macrophage RNA-seq and ATAC-seq experiments, January 2017. URL https://doi.org/10.5281/zenodo.1188300.

JJ Allaire, Yihui Xie, Jonathan McPherson, Javier Luraschi, Kevin Ushey, Aron Atkins, Hadley Wickham, Joe Cheng, Winston Chang, and Richard Iannone. *rmarkdown: Dynamic Documents for R*, 2019. URL https://github.com/rstudio/rmarkdown. R package version 1.18.

A Frankish, GENCODE-consoritum, and P Flicek. GENCODE reference annotation for the human and mouse genomes. *Nucleic Acids Research*, 2018. doi: 10.1093/nar/gky955.

KD Hansen, RA Irizarry, and Wu Z. Removing technical variability in RNA-seq data using condition quantile normalization. *Biostatistics*, 13:204–16, 2012. doi: 10.1093/biostatistics/kxr054.

Johannes Köster and Sven Rahmann. Snakemake—a scalable bioinformatics workflow engine. *Bioinformatics*, 28(19):2520–2522, 08 2012. doi: 10.1093/bioinformatics/bts480.

Charity W Law, Monther Alhamdoosh, Shian Su, Xueyi Dong, Luyi Tian, Gordon K Smyth, and Matthew E Ritchie. RNA-seq analysis is easy as 1-2-3 with limma, glimma and edger. *F1000Res.*, 5(1408):1408, December 2018. doi: 10.12688/f1000research.9005.3. URL https://f1000research.com/articles/5-1408/v3/pdf.

Michael Lawrence, Wolfgang Huber, Hervé Pagès, Patrick Aboyoun, Marc Carlson, Robert Gentleman, Martin T. Morgan, and Vincent J. Carey. Software for computing and annotating genomic ranges. *PLOS Computational Biology*, 9(8):1–10, 08 2013. doi: 10.1371/journal.pcbi.1003118.

Stuart Lee, Dianne Cook, and Michael Lawrence. plyranges: a grammar of genomic data transformation. *Genome Biology*, 20(1):4, Jan 2019. doi: 10.1186/s13059-018-1597-8.

Michael I. Love, Wolfgang Huber, and Simon Anders. Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. *Genome Biology*, 15:550, 2014. doi: 10.1186/s13059-014-0550-8.

Michael I Love, Simon Anders, Vladislav Kim, and Wolfgang Huber. RNA-Seq workflow: gene-level exploratory analysis and differential expression. *F1000Res.*, 4(1070):1070, November 2016. doi: 10.12688/f1000research.7035.2. URL https://f1000research.com/articles/4-1070/v2/pdf.

Michael I Love, Charlotte Soneson, Peter F Hickey, Lisa K Johnson, N Tessa Pierce, Lori Shepherd, Martin Morgan, and Rob Patro. Tximeta: reference sequence checksums for provenance identification in RNA-seq. *bioRxiv*, page 777888, Sep 2019. doi: 10.1101/777888. URL https://www.biorxiv.org/content/10.1101/777888v1.

R Patro, G Duggal, MI Love, RA Irizarry, and C Kingsford. Salmon provides fast and bias-aware quantification of transcript expression. *Nature Methods*, 14:417–419, 2017. doi: 10.1038/nmeth.4197.

R Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria, 2019. URL https://www.R-project.org/.

Lori Shepherd and Martin Morgan. *BiocFileCache: Manage Files Across Sessions*, 2019. R package version 1.10.2.

Mike L. Smith, Andrzej K. Oleś, and Wolfgang Huber. Authoring bioconductor workflows with biocworkflowtools [version 1; referees: 2 approved with reservations]. *F1000Research*, page 431, 2018. doi: 10.12688/f1000research.14399.1.

Gordon K. Smyth. Linear Models and Empirical Bayes Methods for Assessing Differential Expression in Microarray Experiments. *Statistical Applications in Genetics and Molecular Biology*, 3(1), 2004.

Charlotte Soneson, Michael I. Love, and Mark Robinson. Differential analyses for RNA-seq: transcript-level estimates improve gene-level inferences. *F1000Research*, 4, 2015. doi: 10.12688/f1000research.7563.1.

Hadley Wickham, Mara Averick, Jennifer Bryan, Winston Chang, Lucy D'Agostino McGowan, Romain François, Garrett Grolemund, Alex Hayes, Lionel Henry, Jim Hester, Max Kuhn, Thomas Lin Pedersen, Evan Miller, Stephan Milton Bache, Kirill Müller, Jeroen Ooms, David Robinson, Dana Paige Seidel, Vitalie Spinu, Kohske Takahashi, Davis Vaughan, Claus Wilke, Kara Woo, and Hiroaki Yutani. Welcome to the tidyverse. *Journal of Open Source Software*, 4(43):1686, 2019. doi: 10.21105/joss.01686.

Yihui Xie. *Dynamic Documents with R and knitr.* Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition, 2015. URL https://yihui.name/knitr/. ISBN 978-1498716963.

Yihui Xie. *knitr: A General-Purpose Package for Dynamic Report Generation in R*, 2019. URL https://yihui.name/knitr/. R package version 1.26.