R / Bioconductor for Everyone — Exercises

Martin Morgan (mtmorgan@fhcrc.org)
Fred Hutchinson Cancer Research Center
Seattle, WA

30 July 2014

Abstract

This lab provides an introduction to *R / Bioconductor* for high-throughput sequence analysis. It is designed for those who have some but not a lot of familiarity with R and Bioconductor. The first part of the lab focuses on *R* data types, functions, classes, methods, the package and help systems, and the Bioconductor web site. The second part of the lab takes a quick tour of essential packages, classes, and methods for sequence analysis. We will make brief stops at *GenomicRanges*, *Biostrings*, *GenomicFeatures*, *ShortRead*, *Rsamtools*, *rtracklayer*, *AnnotationDbi*, and other packages of interest to participants.

Contents

1 R / Bioconductor **1**

2 Sequence analysis **4**
2.1 Typical data and operations 4
2.2 Large data 6

1 R / Bioconductor

Exercise 1
This exercise uses data describing 128 microarray samples as a basis for exploring R functions. Covariates such as age, sex, type, stage of the disease, etc., are in a data file `pData.csv`*.*

The following command creates a variable `pdataFiles` *that is the location of a comma-separated value ('csv') file to be used in the exercise. A csv file can be created using, e.g., 'Save as...' in spreadsheet software.*

```
pdataFile <- "/home/martin/RBiocForEveryone/pData.csv"
```

Input the csv file using `read.table`*, assigning the input to a variable* `pdata`*. Use* `dim` *to find out the dimensions (number of rows, number of columns) in the object. Are there 128 rows? Use* `names` *or* `colnames` *to list the names of the columns of* `pdata`*. Use* `summary` *to summarize each column of the data. What are the data types of each column in the data frame?*

A data frame is a list of equal length vectors. Select the 'sex' column of the data frame using `[[` *or* `$`*. Pause to explain to your neighbor why this subset operation works. Since a data frame is a list, use* `sapply` *to ask about the class of each column in the data frame. Explain to your neighbor what this produces, and why.*

Use `table` *to summarize the number of males and females in the sample. Consult the help page* `?table` *to figure out additional arguments required to include* `NA` *values in the tabulation.*

The `mol.biol` column summarizes molecular biological attributes of each sample. Use `table` to summarize the different molecular biology levels in the sample. Use `%in%` to create a logical vector of the samples that are either `BCR/ABL` or `NEG`. Subset the original phenotypic data to contain those samples that are `BCR/ABL` or `NEG`.

After subsetting, what are the levels of the `mol.biol` column? Set the levels to be `BCR/ABL` and `NEG`, i.e., the levels in the subset.

One would like covariates to be similar across groups of interest. Use `t.test` to assess whether `BCR/ABL` and `NEG` have individuals with similar age. To do this, use a `formula` that describes the response `age` in terms of the predictor `mol.biol`. If age is not independent of molecular biology, what complications might this introduce into subsequent analysis? Use

Solution: Here we input the data and explore basic properties.

```
pdata <- read.table(pdataFile)
dim(pdata)

## [1] 128  21

names(pdata)

##  [1] "cod"            "diagnosis"      "sex"            "age"            "BT"
##  [6] "remission"      "CR"             "date.cr"        "t.4.11."        "t.9.22."
## [11] "cyto.normal"    "citog"          "mol.biol"       "fusion.protein" "mdr"
## [16] "kinet"          "ccr"            "relapse"        "transplant"     "f.u"
## [21] "date.last.seen"
```

A data frame can be subset as if it were a matrix, or a list of column vectors.

```
head(pdata$sex)                # same as pdata[,"sex"], pdata[["sex"]]

## [1] M M F M M M
## Levels: F M

sapply(pdata, class)

##            cod      diagnosis            sex            age             BT      remission
##       "factor"       "factor"       "factor"      "integer"       "factor"       "factor"
##             CR        date.cr       t.4.11.        t.9.22.    cyto.normal          citog
##       "factor"       "factor"      "logical"      "logical"      "logical"       "factor"
##       mol.biol fusion.protein            mdr          kinet            ccr        relapse
##       "factor"       "factor"       "factor"       "factor"      "logical"      "logical"
##     transplant            f.u date.last.seen
##      "logical"       "factor"       "factor"
```

The number of males and females, including `NA`, is

```
table(pdata$sex, useNA="ifany")

##
##    F    M <NA>
##   42   83    3
```

An alternative version of this uses the `with` function: `with(pdata, table(sex, useNA="ifany"))`.

The `mol.biol` column contains the following samples:

```
with(pdata, table(mol.biol, useNA="ifany"))

## mol.biol
## ALL1/AF4  BCR/ABL E2A/PBX1      NEG   NUP-98  p15/p16
##       10       37        5       74        1        1
```

A logical vector indicating that the corresponding row is either BCR/ABL or NEG is constructed as

```
ridx <- pdata$mol.biol %in% c("BCR/ABL", "NEG")
```

We can get a sense of the number of rows selected via `table` or `sum` (discuss with your neighbor what `sum` does, and why the answer is the same as the number of `TRUE` values in the result of the `table` function).

```
table(ridx)

## ridx
## FALSE  TRUE
##    17   111

sum(ridx)

## [1] 111
```

The original data frame can be subset to contain only BCR/ABL or NEG samples using the logical vector `ridx` that we created.

```
pdata1 <- pdata[ridx,]
```

The levels of each factor reflect the levels in the original object, rather than the levels in the subset object, e.g.,

```
levels(pdata$mol.biol)

## [1] "ALL1/AF4" "BCR/ABL"  "E2A/PBX1" "NEG"      "NUP-98"   "p15/p16"
```

These can be re-coded by updating the new data frame to contain a factor with the desired levels.

```
pdata1$mol.biol <- factor(pdata1$mol.biol)
table(pdata1$mol.biol)

##
## BCR/ABL     NEG
##      37      74
```

To ask whether age differs between molecular biology types, we use a formula `age ~ mol.biol` to describe the relationship ('age as a function of molecular biology') that we wish to test

```
with(pdata1, t.test(age ~ mol.biol))

##
##  Welch Two Sample t-test
##
## data:  age by mol.biol
## t = 4.817, df = 68.53, p-value = 8.401e-06
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##   7.135 17.224
## sample estimates:
## mean in group BCR/ABL     mean in group NEG
##                 40.25                 28.07
```

This summary can be visualize with, e.g., the `boxplot` function

```
## not evaluated
boxplot(age ~ mol.biol, pdata1)
```

Molecular biology seems to be strongly associated with age; individuals in the NEG group are considerably younger than those in the BCR/ABL group. We might wish to include age as a covariate in any subsequent analysis seeking to relate molecular biology to gene expression.

2 Sequence analysis

2.1 Typical data and operations

Exercise 2
The purpose of this exercise is to explore BAM files and the data they contain. We'll extract the sequences and qualities of aligned reads, and draw simple plots to illustrate their properties.

a. *Attach the pasillaBamSubset experiment data package (containing a subset of reads from [1], as described in the package help page). Attach the Rsamtools (for accessing BAM files) and ShortRead (for manipulating short reads) packages.*
b. *Use `ScanBamParam` to create an object that restricts the information extracted from the BAM file. Restrict the information using `readGAlignments` to select reads that align to the plus strand (`isMinusStrand=FALSE`). `readGAlignments` inputs alignment information; also arrange to input sequence and quality information using `what=c('seq', 'qual')`.*
c. *Input the sequences and qualities using `readGAlignments`, providing the file path returned by `untreated1_chr4` (a 'convenience' function from the pasillaBamSubset package) and the ScanBamParam object created in the previous question.*
d. *Use `alphabetByCycle` to summarize nucleotide use by cycle; plot the relationship between cycle and use. Where you expecting this? What is going on?*
e. *Coerce the quality scores to a matrix, and summarize the distribution of qualities across cycle. Is this expected?*

Solution: Here we attach the data and other packages.

```
library(pasillaBamSubset)
library(GenomicRanges)  # readGAlignments
library(ShortRead)  # alphabetByCycle
```

Now establish the *ScanBamParam* object to restrict information extracted from the BAM file.

```
flag <- scanBamFlag(isMinusStrand=FALSE)
param <- ScanBamParam(what=c("seq", "qual"), flag=flag)
```

Input the sequences and qualities using `readGAlignments`.

```
fl <- untreated1_chr4()
res <- readGAlignments(fl, param=param)
```

Summarize nucleotide use by cycle and plot the result (Figure 1, left).

Convert encoding quality scores to a numeric matrix, and summarize these as boxplots, one for each cycle (column) of the matrix (Figure 1, right).

Exercise 3
The purpose of this exercise is to count the number of reads overlapping genes. This is a typical first step in an RNA-seq differential expression analysis.

a. *Annotations (gene models) can come from different sources. We'll use a pre-packaged source here; a later exercise retrieves similar information from a GFF file. Attach the TxDb.Dmelanogaster.UCSC.dm3.ensGene annotation package, and use `exonsBy` to retrieve all exons, grouped by gene. Subset the result to contain just those annotations relevant to chromosome 4.*
b. *Create a character vector of file paths, using the helper functions `untreated1_chr4` and `untreated3_chr4`. Name the vector with an abbreviated name derived from the file name, and use this to invoke `BamFileList`, creating a list of paths to files that are known to be BAM files.*
c. *Use `summarizeOverlaps` to count the number of reads in each region of interest. The RNA-seq protocol was not strand aware, so choose `ignore.strand=TRUE`. The object returned by `summarizeOverlaps` is an instance of class SummarizedExperiment. Explore it, and the counts that have been generate.*

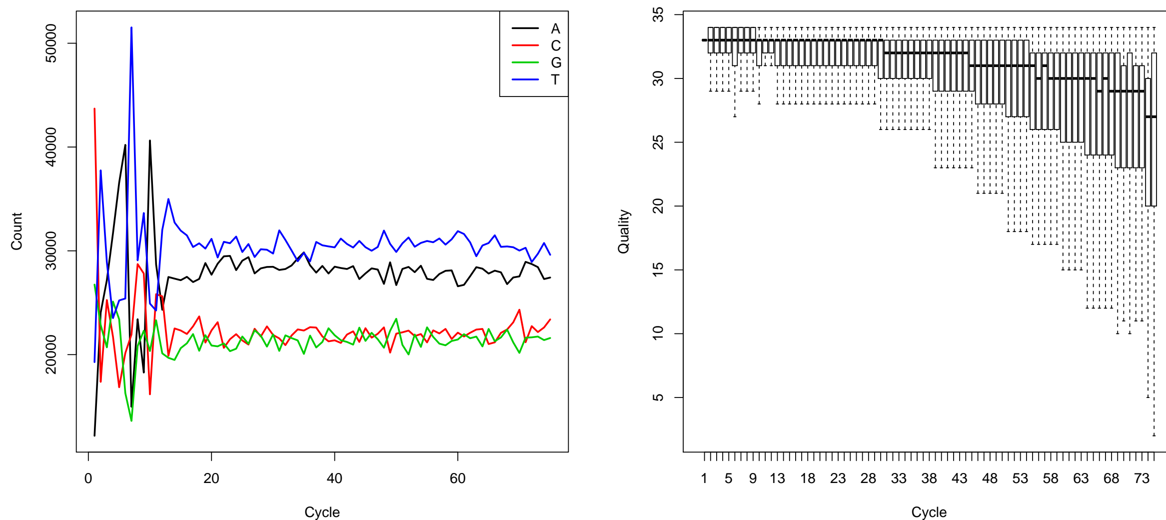

Figure 1: Nucleotide use (left) and base quality (right) in aligned reads.

d. *Plot the counts in the two samples, perhaps transforming the counts with* `asinh` *(a log-like transform that deals with 0's better).*
e. *As an additional exercise, use the rtracklayer package to retrieve and parse a GFF file containing similar annotations. How would this source of annotation tie into the work flow we have just performed?*

Solution: Load the annotation package, extract exons grouped by gene, and select just those genes on chromosome 4.

```
library(TxDb.Dmelanogaster.UCSC.dm3.ensGene)  # genome coordinates
exByGn <- exonsBy(TxDb.Dmelanogaster.UCSC.dm3.ensGene, "gene")
chr4 <- exByGn[ all(seqnames(exByGn) == "chr4") ]
```

Create a *BamFileList* pointing to the BAM files we want to count.

```
fls <- c(untreated1_chr4(), untreated3_chr4())
names(fls) <- sub("_chr4.bam", "", basename(fls))
bfl <- BamFileList(fls)
```

Count the number of reads overlapping each gene using `summarizeOverlaps`, remembering to the ignore strand to which the read aligns (why?).

```
counts <- summarizeOverlaps(chr4, bfl, ignore.strand=TRUE)
head(assay(counts))

##             untreated1 untreated3
## FBgn0002521        669        409
## FBgn0004607         13         12
## FBgn0004859        382        198
## FBgn0005558          8         22
## FBgn0005561          7         11
## FBgn0005666        491        276
```

Plot the results (Figure 2). Why are the points generally below the diagonal? Why do they have a funnel shape, with lots of variability between samples at low counts but very predictable numbers at high counts?

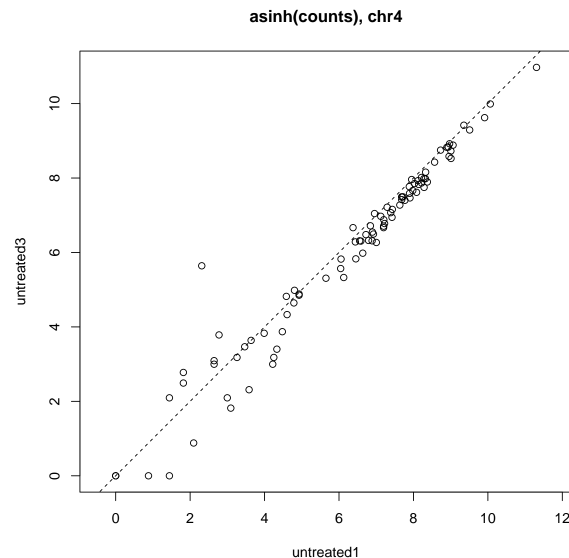

Figure 2: Count of reads overlapping genes on chr4.

The following illustrates an alternative source of annotation information. The annotations are in a GTF file on the Ensembl web site. We download them to our local disk, then use *rtracklayer*'s `import` to parse the file to a *GRanges* instance. We subset this to contain relevant information, then split the *GRanges* into a *GRangesList*, based on the `gene_id` column extracted from the GFF file. This could then be used in `summarizeOverlaps`, as above and without further change.

```
library(rtracklayer)                    # import gff
fl <- paste0("ftp://ftp.ensembl.org/pub/release-62/", "gtf/drosophila_melanogaster/",
             "Drosophila_melanogaster.BDGP5.25.62.gtf.gz")
gffFile <- file.path(tempdir(), basename(fl))
download.file(fl, gffFile)
gff0 <- import(gffFile)
idx <- gff0$source == "protein_coding" & gff0$type == "exon" & seqnames(gff0) == "4"
gff1 <- gff0[idx]
chr4.gff <- split(gff1, mcols(gff1)$gene_id)
```

2.2 Large data

Counting overlaps represents a typical operation on BAM files. It involves accessing several large files for moderate amounts of data, then summarizing that data to a relatively compact representation (a table of counts). We sketch how to efficiently perform this type of operation in *R*. To do so we suppose a counting function that accepts a `GAlignments` object representing the data, and a `GRanges` (or `GRangesList`) object representing the regions of interest over which counts are required.

```
counter <-
    function(aln, roi)
{
    strand(aln) <- "*"                  # strand-neutral protocol
    hits <- findOverlaps(aln, roi)
    keep <- which(countQueryHits(hits) == 1)
    cnts <- countSubjectHits(hits[queryHits(hits) %in% keep])
```

```
    setNames(cnts, names(roi))
}
```

This function is written using efficient *R* code. A first pass at using the counter might use `sapply` and a helper function to iterate over each file

```
countFile <-
    function(fl, roi)
{
    open(fl); on.exit(close(fl))
    aln <- readGAlignments(fl)
    counter(aln, roi)
}
count0 <- sapply(bfl, countFile, chr4)
head(count0)

##             untreated1 untreated3
## FBgn0002521        669        409
## FBgn0004607         13         12
## FBgn0004859        382        198
## FBgn0005558          8         22
## FBgn0005561          7         11
## FBgn0005666        491        276
```

BAM files can be large, so it might pay to iterate through each in 'chunks' that are large but do not consume all available memory. The pattern might look like

```
countInChunks <-
    function(fl, roi)
{
    yieldSize(fl) <- 1000000            # chunks of size 1 million
    open(fl); on.exit(close(fl))
    count <- integer(length(range))     # initial count vector
    while (length(aln <- readGAlignments(fl)))
        count <- count + counter(aln, roi)
    count
}
count1 <- sapply(bfl, countInChunks, chr4)
identical(count0, count1)

## [1] TRUE
```

Chunking does not usually add a significant performance cost, but requires an algorithm where successive chunks can be calculated independently (as with counting overlaps) and aggregated easily (here simply adding successive count vectors).

Chunking allows memory use to be adjusted to different scenerios, and in particular opens the door to iterating through large files in parallel, with each processor associated with one file. This is easily implemented in *R*, using the `parallel::mclapply` function (which acts like `lapply` but assigns elements of its first argument to different processors), and `base::simplify2Array`.

```
library(parallel)
options(mc.cores=detectCores())         # use all cores for parallel evaluation
mcsapply <- function(...) simplify2array(mclapply(...))
count2 <- mcsapply(bfl, countInChunks, chr4)
identical(count0, count2)

## [1] TRUE
```

This strategy is implemented in `summarizeOverlaps`.

References

[1] A. N. Brooks, L. Yang, M. O. Duff, K. D. Hansen, J. W. Park, S. Dudoit, S. E. Brenner, and B. R. Graveley. Conservation of an RNA regulatory map between Drosophila and mammals. *Genome Research*, pages 193–202, 2011. URL: http://genome.cshlp.org/cgi/doi/10.1101/gr.108662.110, doi:10.1101/gr.108662.110.