CB2-101: R for Bioinformatics

Malay (malay@uab.edu)

November 14, 2016

Contents

1 Some useful resources

R was a popular tool for analysis of microarray data. Now it is mostly used in Bioinformatics for analysis of next-gen sequence data. It is not very popular as a general purpose Bioinformatics tool. There are a bunch of special packages distributed under the name “BioConductor” (http://www.bioconductor.org/) that are related to biological data analysis using R.

- Bioinformatics using R
  1. A little Book of R for Bioinformatics (http://a-little-book-of-r-for-bioinformatics.readthedocs.org/en/latest/).
- Learning BioConductor
  1. BioConductor help section contains exhaustive lists of conferences (http://www.bioconductor.org/help/course-materials/). The course materials of these conferences are very good.
  2. A nice intermediate level guide to R and BioConductor: http://www.bioconductor.org/help/course-materials/2013/SeattleMay2013/IntermediateSequenceAnalysis2013.pdf.
  3. A somewhat scattered introduction to NGS data analysis using R and BioConductor: http://manuals.bioinformatics.ucr.edu/home/ht-seq

You can see there is sharp drop of quality below score 20. This is why Phred 20 is a good cutoff score. This actually (20 + 33) = 53 which `5` in ascii.

1.1 BAM or SAM format

The FASTQ files are aligned against a reference genome using a software like BWA (http://bio-bwa.sourceforge.net/). The resulting alignment format is a BAM or SAM files. BAM files are binary, SAM files are plain text. The software for interconversion and analysis of these files are mainly `samtools` (http://www.htslib.org/). A small example BAM files comes along with `Rsamtools` package. Sam file format specification can be found here http://samtools.github.io/hts-specs/SAMv1.pdf.

1.2 VCF

Once the alignment BAM files have been generated, a variant caller like GATK (https://www.broadinstitute.org/gatk/) is used to find the variants in the file. The resulting file is called VCF. The specification can be found here (http://samtools.github.io/hts-specs/VCFv4.2.pdf). A sample VCF line is given below:

```
chr1    873762  .       T   G   [CLIPPED] GT:AD:DP:GQ:PL    0/1:173,141:282:99:255,0,255
chr1    877664  rs3828047   A   G   [CLIPPED] GT:AD:DP:GQ:PL    1/1:0,105:94:99:255,255,0
chr1    899282  rs28548431  C   T   [CLIPPED] GT:AD:DP:GQ:PL    0/1:1,3:4:25.92:103,0,26
```

Once the variant is called they are annotated using variant annotation tools like SnpEff (http://snpeff.sourceforge.net/) or Annovar (http://www.openbioinformatics.org/annovar/) or `VariantAnnotation` package. –> # Installing BioConductor packages

All bioconductor packages are installed using the following commands:

```r
source("http://bioconductor.org/biocLite.R")
biocLite("packagename")
```

Where, `packagename` is the name of your BioConductor package.

2 Where to start ?

BioConductor is a jumbled mess of hundreds of packages. And a problem for the beginners is to know where to start and which packages to use. I suggest you start with the workflows page of BC (http://bioconductor.org/help/workflows/). Look at the examples and find out what packages are used and then go and dig for more information about those packages.

3 A simple example

Lets start with a simple example. Remember, we calculated the average protein length of *E. coli* in our Linux problem set. Let's solve this using BC. The package that we need is `Biostrings`. Let's install the package.

```r
source("http://bioconductor.org/biocLite.R")
biocLite("Biostrings")
```

Once the package is installed. We have to now load it.

```r
suppressPackageStartupMessages( library("Biostrings") )
```

You can see an overview of what `Biostrings` package has to offer.

```r
browseVignettes("Biostrings")
```

You can now get a quick overview by clicking on "Biostrings quick overview" PDF. By looking at the quick overview, we find that there is a function in `Biostrings` that can read the sequence: `readAAStringSet()`.

```
# Just to get the long line to wrap correctly
url <- paste("ftp://ftp.ncbi.nlm.nih.gov/genomes/archive/old_refseq/Bacteria/Escherichia_coli_K_12_subs
faa <- readAAStringSet(url)
```

We can get the average length now:

```
av.length <- sum( width(faa) )/length(faa)
av.length
```

```
## [1] 316.8587
```

4 Some basic objects in BioConductor

4.1 IRanges

`IRanges` represents orders indices.

```
library(IRanges)
r <- IRanges(start=c(1,3,12,10),end=c(4,5,25,19))
r
```

```
## IRanges of length 4
##     start end width
## [1]     1   4     4
## [2]     3   5     3
## [3]    12  25    14
## [4]    10  19    10
```

4.1.1 Simple operations on IRanges

```
length(r)
```

```
## [1] 4
```

```
start(r)
```

```
## [1]  1  3 12 10
```

```
end(r)
```

```
## [1]  4  5 25 19
```

```
width(r)
```

```
## [1]  4  3 14 10
```

```
r[1:2]
```

```
## IRanges of length 2
##     start end width
## [1]     1   4     4
## [2]     3   5     3
```

```
range(r)
```

```
## IRanges of length 1
##     start end width
## [1]     1  25    25
```

```
reduce(r)
```

```
## IRanges of length 2
##     start end width
## [1]     1   5     5
## [2]    10  25    16
```

```
disjoin(r)
```

```
## IRanges of length 6
##     start end width
## [1]     1   2     2
## [2]     3   4     2
## [3]     5   5     1
## [4]    10  11     2
## [5]    12  19     8
## [6]    20  25     6
```

```
coverage(r)
```

```
## integer-Rle of length 25 with 7 runs
##   Lengths: 2 2 1 4 2 8 6
##   Values : 1 2 1 0 1 2 1
```

4.1.2 Getting the flanking region

```
flank(r, 1, both=T,start=T)
```

```
## IRanges of length 4
##     start end width
## [1]     0   1     2
## [2]     2   3     2
## [3]    11  12     2
## [4]     9  10     2
```

4.1.3 Set operations on ranges

```
r2 <- IRanges(start=c(7,8,14),end=c(11,16,18))
union(r,r2)
```

```
## IRanges of length 2
##     start end width
## [1]     1   5     5
## [2]     7  25    19
```

```
intersect(r,r2)
```

```
## IRanges of length 1
##     start end width
## [1]    10  18     9
```

```
setdiff(r,r2)
```

```
## IRanges of length 2
##     start end width
## [1]     1   5     5
## [2]    19  25     7
```

4.2 Run length Encoding (RLE)

```
x<- Rle(c(1,1,2,2,2))
length(x)
```

```
## [1] 5
```

```
start(x)
```

```
## [1] 1 3
```

```
end(x)
```

```
## [1] 2 5
```

```
width(x)
```

```
## [1] 2 3
```

```
nrun(x)
```

```
## [1] 2
```

```
runLength(x)
```

```
## [1] 2 3
```

4.3 GenomicRanges

There are 3 classes in this package: `GRanges`, `GRangeList`, `GappedAlignments`.

4.3.1 GRanges

```
library(GenomicRanges)
```

```
## Loading required package: GenomeInfoDb
```

```
gr <- GRanges(seqnames= Rle(c("chr1","chr2"),c(2,3)),
             ranges = IRanges (1:5, end= 6:10),
             strand = Rle(strand(c("-","+","+","-","+"))),
             score=1:5, GC=seq(1,0,length=5))
gr
```

```
## GRanges object with 5 ranges and 2 metadata columns:
##       seqnames    ranges strand |     score        GC
##          <Rle> <IRanges>  <Rle> | <integer> <numeric>
##   [1]     chr1   [1,  6]      - |         1         1
##   [2]     chr1   [2,  7]      + |         2      0.75
##   [3]     chr2   [3,  8]      + |         3       0.5
##   [4]     chr2   [4,  9]      - |         4      0.25
##   [5]     chr2   [5, 10]      + |         5         0
##   -------
##   seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

4.3.1.1 Access elements of GRanges

```r
length(gr)
```

```
## [1] 5
```

```r
seqnames(gr)
```

```
## factor-Rle of length 5 with 2 runs
##   Lengths:    2    3
##   Values : chr1 chr2
## Levels(2): chr1 chr2
```

```r
start(gr)
```

```
## [1] 1 2 3 4 5
```

```r
end(gr)
```

```
## [1]  6  7  8  9 10
```

```r
ranges(gr)
```

```
## IRanges of length 5
##     start end width
## [1]     1   6     6
## [2]     2   7     6
## [3]     3   8     6
## [4]     4   9     6
## [5]     5  10     6
```

```r
strand(gr)
```

```
## factor-Rle of length 5 with 4 runs
##   Lengths: 1 2 1 1
##   Values : - + - +
## Levels(3): + - *
```

All other fields besides `seqnames`, `range` and `strands` need to be accessed by `elementMetadata` function.

```r
elementMetadata(gr)
```

```
## DataFrame with 5 rows and 2 columns
##       score        GC
##   <integer> <numeric>
## 1         1      1.00
## 2         2      0.75
## 3         3      0.50
## 4         4      0.25
## 5         5      0.00
```

4.3.2 GRangesList

It's a list of `GRanges` objects.

```r
GRangesList (gr, gr)
```

```
## GRangesList object of length 2:
## [[1]]
## GRanges object with 5 ranges and 2 metadata columns:
##       seqnames    ranges strand |     score        GC
##          <Rle> <IRanges>  <Rle> | <integer> <numeric>
##   [1]     chr1   [1,  6]      - |         1         1
##   [2]     chr1   [2,  7]      + |         2      0.75
##   [3]     chr2   [3,  8]      + |         3       0.5
##   [4]     chr2   [4,  9]      - |         4      0.25
##   [5]     chr2   [5, 10]      + |         5         0
##
## [[2]]
## GRanges object with 5 ranges and 2 metadata columns:
##       seqnames  ranges strand | score   GC
##   [1]     chr1 [1,  6]      - |     1    1
##   [2]     chr1 [2,  7]      + |     2 0.75
##   [3]     chr2 [3,  8]      + |     3  0.5
##   [4]     chr2 [4,  9]      - |     4 0.25
##   [5]     chr2 [5, 10]      + |     5    0
##
## -------
## seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

4.3.3 GappedAlignments

Used for parsing BAM files.

4.4 BSgenome

`BSgenome` is the actual genome sequences distributed in a R package. This packages can be pretty big. For human this file is about 1.7G in size. For this course, we will not use it anymore.

5 Annotation database

There are two types of annotation databases in BC. Organism-specific gene level databases are names as `org.XX.XXX.db`. For e.g., `org.Hs.eg.db`. This is human entrez gene database. There are also metapackaces (not for all organisms) that pull data from may different sources. `Homo.sapiens` is one such databases. Let's use this database.

```r
suppressPackageStartupMessages(library("Homo.sapiens"))
columns(Homo.sapiens)
```

```
##  [1] "ACCNUM"       "ALIAS"        "CDSCHROM"     "CDSEND"
##  [5] "CDSID"        "CDSNAME"      "CDSSTART"     "CDSSTRAND"
##  [9] "DEFINITION"   "ENSEMBL"      "ENSEMBLPROT"  "ENSEMBLTRANS"
## [13] "ENTREZID"     "ENZYME"       "EVIDENCE"     "EVIDENCEALL"
## [17] "EXONCHROM"    "EXONEND"      "EXONID"       "EXONNAME"
## [21] "EXONRANK"     "EXONSTART"    "EXONSTRAND"   "GENEID"
## [25] "GENENAME"     "GO"           "GOALL"        "GOID"
## [29] "IPI"          "MAP"          "OMIM"         "ONTOLOGY"
## [33] "ONTOLOGYALL"  "PATH"         "PFAM"         "PMID"
## [37] "PROSITE"      "REFSEQ"       "SYMBOL"       "TERM"
## [41] "TXCHROM"      "TXEND"        "TXID"         "TXNAME"
## [45] "TXSTART"      "TXSTRAND"     "TXTYPE"       "UCSCKG"
## [49] "UNIGENE"      "UNIPROT"
```

Only some of these columns can be use to retrieve data. To find what columns can be used

```
keytypes(Homo.sapiens)
```

```
##  [1] "ACCNUM"       "ALIAS"        "CDSID"        "CDSNAME"
##  [5] "DEFINITION"   "ENSEMBL"      "ENSEMBLPROT"  "ENSEMBLTRANS"
##  [9] "ENTREZID"     "ENZYME"       "EVIDENCE"     "EVIDENCEALL"
## [13] "EXONID"       "EXONNAME"     "GENEID"       "GENENAME"
## [17] "GO"           "GOALL"        "GOID"         "IPI"
## [21] "MAP"          "OMIM"         "ONTOLOGY"     "ONTOLOGYALL"
## [25] "PATH"         "PFAM"         "PMID"         "PROSITE"
## [29] "REFSEQ"       "SYMBOL"       "TERM"         "TXID"
## [33] "TXNAME"       "UCSCKG"       "UNIGENE"      "UNIPROT"
```

To extract data we need the "keys" corresponding to a "keytype". For example the `SYMBOL` keytypes stores the gene name and surprisingly `GENENAME` actually contains a description of gene. We can show the partial list of these genes.

```
genenames<-(keys(Homo.sapiens,keytype="SYMBOL"))
```

There are altogether 60083 genes in this database. We can now use `genenames` as keys to get the genes and their longer name for the database.

```
gene.list <-select(Homo.sapiens,keys=genenames,columns=c("SYMBOL","GENENAME"),keytype="SYMBOL")
```

```
## 'select()' returned 1:many mapping between keys and columns
```

```
head(gene.list)
```

```
##   SYMBOL                           GENENAME
## 1   A1BG             alpha-1-B glycoprotein
## 2    A2M              alpha-2-macroglobulin
## 3  A2MP1 alpha-2-macroglobulin pseudogene 1
## 4   NAT1              N-acetyltransferase 1
## 5   NAT2              N-acetyltransferase 2
## 6   NATP     N-acetyltransferase pseudogene
```

Let's do something interesting. Let plot the number of genes per chromosomes.

```
# Old bioconductor gene.df <-
# select(Homo.sapiens,keys=genenames,columns=c('CHROM','CHRLOC','CHRLOCEND'),keytype='SYMBOL')
gene.df <- select(Homo.sapiens, keys = genenames, columns = c("TXCHROM"), keytype = "SYMBOL")
```

```
## 'select()' returned 1:many mapping between keys and columns
```

```
head(gene.df)
```

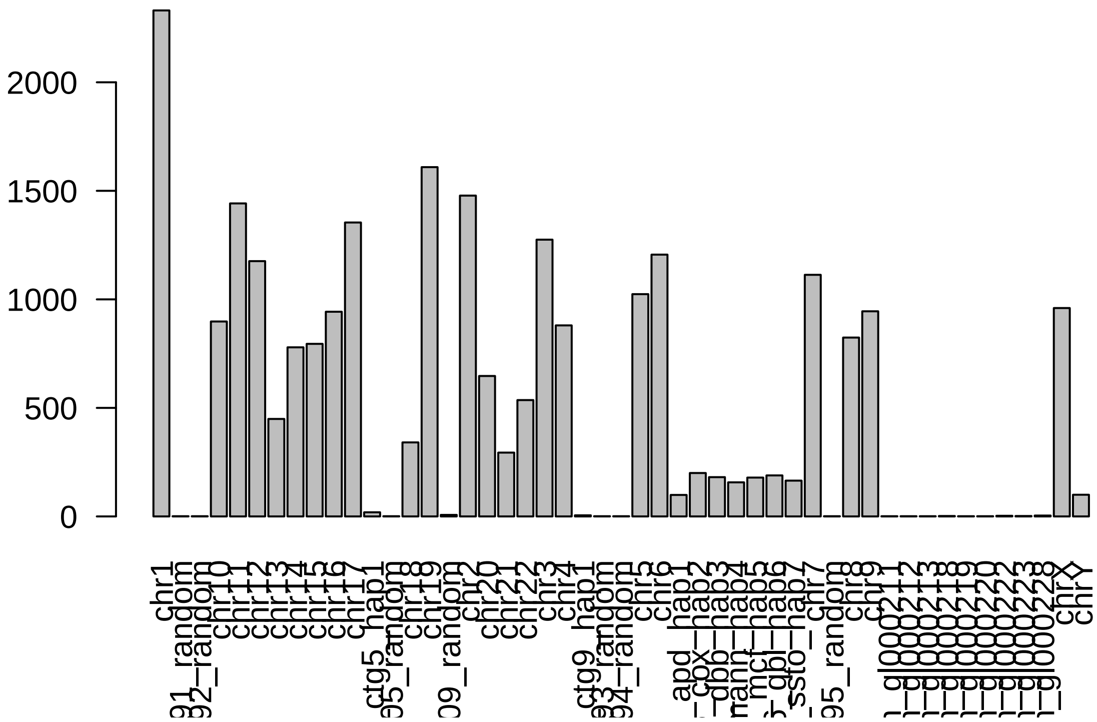
```
##   SYMBOL TXCHROM
## 1   A1BG   chr19
## 2    A2M   chr12
## 3  A2MP1   chr12
## 4   NAT1    chr8
## 5   NAT2    chr8
## 6   NATP    <NA>
```

```r
# Let's extract the SYMBOL and CHR is a separate dataframe.
gene.uniq <- data.frame(symbol = gene.df$SYMBOL, chr = gene.df$TXCHROM)

# Let's remove the duplicated lines.
gene.uniq <- gene.uniq[order(gene.uniq$symbol), ]
gene.uniq <- gene.uniq[!duplicated(gene.uniq), ]
head(gene.uniq)
```

```
##          symbol  chr
## 38980 1060P11.3 <NA>
## 23563   3.8-1.2 <NA>
## 23564   3.8-1.3 <NA>
## 23565   3.8-1.4 <NA>
## 23566   3.8-1.5 <NA>
## 24362   5-HT3C2 <NA>
```

```r
gene.freq <- table(gene.uniq$chr)
barplot(table(gene.uniq$chr), las = 2)
```



## Number of genes per chromosome

One of hypothesis that we can check whether the number of genes are correlated with the length of the chromosome. To get the length of the chromosome, we need to load another package in R `GenomicFeatures`.

```r
suppressPackageStartupMessages(library("GenomicFeatures"))
chr.info <- getChromInfoFromUCSC("hg19")
```

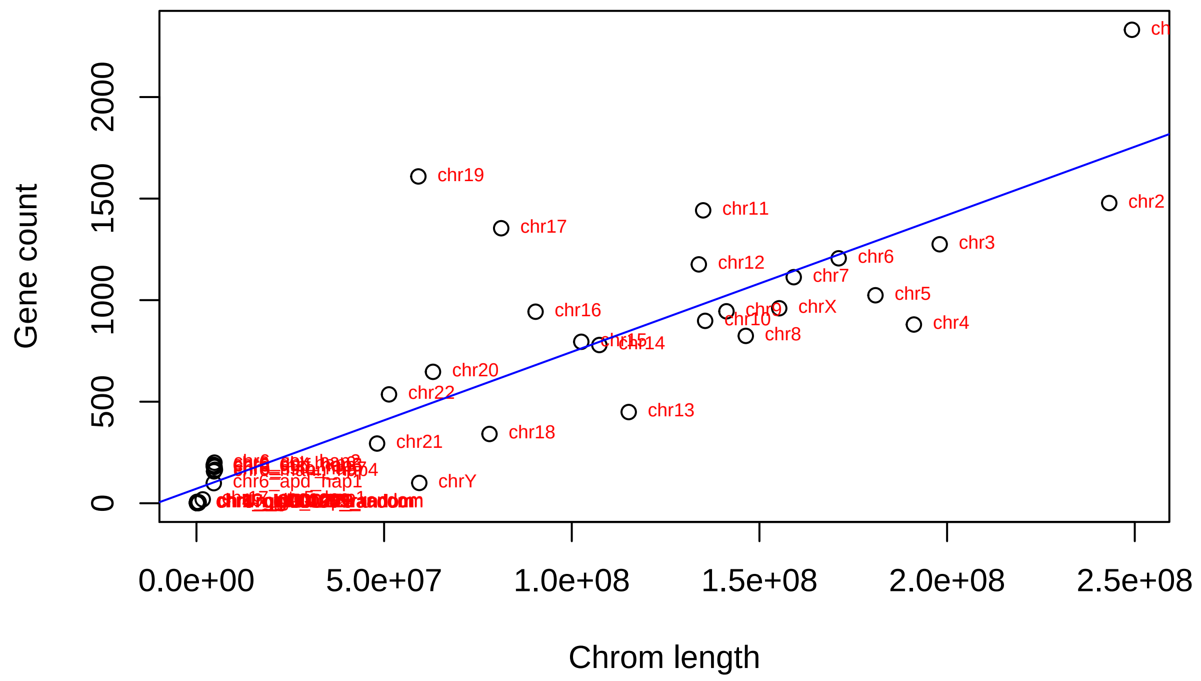
```
## Download and preprocess the 'chrominfo' data frame ...
## OK
```

```r
head(chr.info)
```

```
##   chrom    length
## 1  chr1 249250621
## 2  chr2 243199373
## 3  chr3 198022430
## 4  chr4 191154276
## 5  chr5 180915260
## 6  chr6 171115067
```

```r
# Convert our frequency table into data frame
gene.freq <- data.frame(gene.freq)
names(gene.freq) <- c("chr","freq")

# We need to convert the names of the chr column
#gene.freq$chr <- paste('chr',gene.freq$chr,sep="")
merged.data <- merge(gene.freq,chr.info,by.x="chr",by.y="chrom")
plot(merged.data$length,merged.data$freq,xlab="Chrom length",ylab="Gene count")
text(merged.data$length, merged.data$freq, merged.data$chr, cex=0.6, pos=4, col="red")
abline(lm(merged.data$freq~merged.data$length),col="blue")
```



5.1 What is the mutation frequency of P53 gene in normal human population

For this problem we first have to find the location of the P53 gene in human annotation database.

```
library(Homo.sapiens)
loc <- select(Homo.sapiens,keys="TP53",columns=c("SYMBOL","CHR","CHRLOC","CHRLOCEND"),keytype="SYMBOL")
```

```
## 'select()' returned 1:1 mapping between keys and columns
```

We see that TP53 gene is on chromosome 17 in location 7571720:7590868. We will download this portion of the variation from 1000 genome data using `tabix`. Install `tabix` on your system.

Once `tabix` is installed. We can download this portion of the file using the following command.

```
tabix -fh ftp://ftp-trace.ncbi.nih.gov/1000genomes/ftp/release/20110521/\
ALL.chr17.phase1_release_v3.20101123.snps_indels_svs.genotypes.vcf.gz \
17:7571720-7590868 >p53.vcf
```

Let's read the VCF file in R.

```
library(VariantAnnotation)
vcf <- readVcf("p53.vcf","hg19")
```

We will now locate variant using the txdb package.

```
library("TxDb.Hsapiens.UCSC.hg19.knownGene")
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
txdb <- renameSeqlevels(txdb, gsub("chr","",seqlevels(txdb)))
txdb <- keepSeqlevels(txdb,"17")
all <- locateVariants(vcf,txdb, AllVariants())
table(mcols(all)$LOCATION)
```

Looks like there are 162 variants in the coding regions in the 1000K sample. I will leave it to you to investigate this further.