MCB536 Lecture 16 (Part 1): Genomic Data Analysis in R

Gavin Ha

11/22/2020

`GenomicRanges` Object to Store Genomic Data

Genomic data is often described using chromosomes and coordinates. A locus can be a single base position or a region that includes a start and end coordinate. In R, there is a Bioconductor package called `GenomicRanges` that stores this in a convenient structure for efficient querying using routine operations. `GRanges` object class is in which genomic data will be stored. We will demonstrate the most common operation, `findOverlaps`, to determine intersecting positions or regions in the genome.

In this tutorial, we will work with The Cancer Genome Atlas (TCGA) data for primary breast cancer patient samples. Specifically, these are segmentation data used for copy number alteration analysis. See Lecture 16: Slide 47.

0. Load the `GenomicRanges` Bioconductor package

```r
#BiocManager::install(GenomicRanges")
library(GenomicRanges)
```

1. Create a GRanges object.

A `GRanges` object must contain an attribute called `seqnames` to represent chromosomes and `ranges` attribute to represent the `start` and `end` coordinates. The range is 1-index-based (as opposed to 0-index), The `start` and `end` can be the same value if it is a single base-pair.

```r
myGRange <- GRanges(seqnames = "17",
                    ranges = IRanges(start = 37844393, end = 37844393))
```

2. Load Genomic Data From A File

There are numerous text file formats for representing genomic data and some of these were discussed in Lecture 16. Here, we will show you that a `GRanges` can be easily created from any text file that contains delimited columns specifying genomic coorindates.

2.1 SEG format

SEGment Data (http://software.broadinstitute.org/software/igv/SEG) format is tab-delimited and a flexible way to define any genomic data.

There are 4 required columns:

1. Name
2. Chromosome
3. Start Coordinate
4. End Coordinate

This is similar to the BED file format but with the additional requirement for *Name* as the first column.

a. Load the SEG file containing the segments into a `data.frame` object.

```
segs <- read.table("BRCA.genome_wide_snp_6_broad_Level_3_scna.seg", header = TRUE)
```

Small processing of this file to correct a few legacy hacks. We need to change chromosome 23 to chromosome X.

```
str(segs) # show the class type for each column
```

```
## 'data.frame':    284458 obs. of  6 variables:
##  $ Sample       : chr  "TCGA-3C-AAAU-10A-01D-A41E-01" "TCGA-3C-AAAU-10A-01D-A41E-01" "TCGA-3C-AAAU-10A
##  $ Chromosome   : int  1 1 1 1 1 1 1 1 1 2 ...
##  $ Start        : int  3218610 95676511 95680124 167057495 167059760 181603120 181610685 201474400 20
##  $ End          : int  95674710 95676518 167057183 167059336 181602002 181609567 201473647 201474544 1
##  $ Num_Probes   : int  53225 2 24886 3 9213 6 12002 2 29781 30300 ...
##  $ Segment_Mean: num  0.0055 -1.6636 0.0053 -1.0999 -0.0008 ...
```

```
mode(segs$Chromosome) <- "character" # change the class of the chromosome to character
segs[segs$Chromosome == 23, "Chromosome"] <- "X"
```

b. Convert the `data.frame` object into a `GRanges`.

You can use the `as()` function, as long as the 3 required columns are present. It is also flexible how the columns are named. For example, the column can be `Start`, `start`, `Chr`, `chr`, `Chromosome`, `End`, `Stop`, etc.

```
segs.gr <- as(segs, "GRanges")
segs.gr
```

```
## GRanges object with 284458 ranges and 3 metadata columns:
##            seqnames              ranges strand |                 Sample
##               <Rle>           <IRanges>  <Rle> |            <character>
##        [1]        1    3218610-95674710      * | TCGA-3C-AAAU-10A-01D..
##        [2]        1   95676511-95676518      * | TCGA-3C-AAAU-10A-01D..
##        [3]        1  95680124-167057183      * | TCGA-3C-AAAU-10A-01D..
##        [4]        1 167057495-167059336      * | TCGA-3C-AAAU-10A-01D..
##        [5]        1 167059760-181602002      * | TCGA-3C-AAAU-10A-01D..
##        ...      ...                 ...    ... .                    ...
##   [284454]       19      284018-58878226      * | TCGA-Z7-A8R6-01A-11D..
##   [284455]       20      455764-62219837      * | TCGA-Z7-A8R6-01A-11D..
##   [284456]       21    15347621-47678774      * | TCGA-Z7-A8R6-01A-11D..
##   [284457]       22    17423930-49331012      * | TCGA-Z7-A8R6-01A-11D..
##   [284458]        X    3157107-154905589      * | TCGA-Z7-A8R6-01A-11D..
##            Num_Probes Segment_Mean
##             <integer>    <numeric>
##        [1]      53225       0.0055
##        [2]          2      -1.6636
##        [3]      24886       0.0053
##          [4]          3      -1.0999
##          [5]       9213      -0.0008
##          ...        ...          ...
##     [284454]      23950      -0.1170
##     [284455]      37283       0.3435
##     [284456]      20582      -0.1117
##     [284457]      16927      -0.1231
##     [284458]      63797       0.0014
##   -------
##   seqinfo: 23 sequences from an unspecified genome; no seqlengths
```

3. Operations and features of GenomicRanges

Some of the most useful features of `GRanges` object is the fast and easy methods for determining overlaps between sets of ranges. Here, we will describe examples using some of the common functions.

3.1 Tiling the genome

Often we would like to *find* or *count* events overlapping regions in the genome. In an unbiased fashion, we could do this genome-wide by dividing the genome into tiles/windows/bins. We will use the `tileGenome()` for this task, which requires three arguments: length of the chromosomes, number of tiles and the size of each tile.

a. We need the lengths of the chromosomes in the human genome.

We need to load human genome information for build `hg19`. Since there are non-standard chromosomes, we only want to keep the standard chromosomes using `keepStandardChromosomes`. Then, since our `segs` data uses `NCBI` chromosome naming convention (i.e. `1` instead of `chr1`), we need set the `seqlevelStyle`.

```r
seqinfo <- Seqinfo(genome = "hg19")
seqinfo <- keepStandardChromosomes(seqinfo)
seqlevelsStyle(seqinfo) <- "NCBI"
seqinfo
```

```
## Seqinfo object with 25 sequences (1 circular) from 2 genomes (GRCh37.p13, hg19):
##   seqnames seqlengths isCircular     genome
##   1         249250621      FALSE GRCh37.p13
##   2         243199373      FALSE GRCh37.p13
##   3         198022430      FALSE GRCh37.p13
##   4         191154276      FALSE GRCh37.p13
##   5         180915260      FALSE GRCh37.p13
##   ...             ...        ...        ...
##   21         48129895      FALSE GRCh37.p13
##   22         51304566      FALSE GRCh37.p13
##   X         155270560      FALSE GRCh37.p13
##   Y          59373566      FALSE GRCh37.p13
##   chrM          16571       TRUE       hg19
```

b. Split the genome into 500kb tiles or windows.

```
slen <- seqlengths(seqinfo) # get the length of the chromosomes
tileWidth <- 500000 # tile size of 500kb
tiles <- tileGenome(seqlengths = slen, tilewidth = tileWidth,
                    cut.last.tile.in.chrom = TRUE)
tiles
```

```
## GRanges object with 6207 ranges and 0 metadata columns:
##          seqnames            ranges strand
##             <Rle>         <IRanges>  <Rle>
##      [1]        1          1-500000      *
##      [2]        1    500001-1000000      *
##      [3]        1   1000001-1500000      *
##      [4]        1   1500001-2000000      *
##      [5]        1   2000001-2500000      *
##      ...      ...               ...    ...
##   [6203]        Y 57500001-58000000      *
##   [6204]        Y 58000001-58500000      *
##   [6205]        Y 58500001-59000000      *
##   [6206]        Y 59000001-59373566      *
##   [6207]     chrM           1-16571      *
##   -------
##   seqinfo: 25 sequences from an unspecified genome
```

3.2 Finding overlap of ranges

One of the most useful features of `GenomicRanges` is to simply identify the ranges that overlap between two `GRanges` objects. The `findOverlaps` function is a basic method in the `GRanges` class for finding the overlaps of the elements that overlap between two `GRanges`. The argmuents `query` for your main `tiles.subset` and `subject` for the `segs.gr`. The `type` argument describes the type of overlap, such as `any`, `within`, `start`, `end`, `equal`, and there are additional arguments for criteria for overlap such as `minoverlap` size.

For this example, let's find which copy number alteration segments from `segs.gr` overlap in *any* way with our ranges in `tiles.subset` (`17:35000000-37000000`).

```
tiles.subset <- tiles[5082:5084]
```

a. Find the overlap between segs.gr and tiles.subset.

We will use the function `findOverlaps` to identify overlapping elements in `segs.gr` and `tiles.subset`. For the criteria of any overlap, we set `type = "any"`.

```
hits1 <- findOverlaps(query = tiles.subset, subject = segs.gr, type = "any")
hits1
```

```
## Hits object with 6969 hits and 0 metadata columns:
##          queryHits subjectHits
##          <integer>   <integer>
##      [1]         1          57
##      [2]         1         315
##      [3]         1         453
##      [4]         1         668
##      [5]         1         669
##      ...       ...         ...
##   [6965]         3      283635
##   [6966]         3      283699
##   [6967]         3      283764
##   [6968]         3      284193
##   [6969]         3      284446
##   -------
##   queryLength: 3 / subjectLength: 284458
```

This returns a `Hits` object that indicate the indices of the elements in each object that overlap.

b. Extract the overlapping elements in `segs.gr`.

Let's look at some of the segments in `segs.gr` (`subject`) that overlap the first tile in `tiles.subset` (`query`) at `17:34500001-35000000`.

```
tiles.subset.overlap.ind <- queryHits(hits1)[1]
tiles.subset[tiles.subset.overlap.ind] # this is the first tile
```

```
## GRanges object with 1 range and 0 metadata columns:
##       seqnames            ranges strand
##          <Rle>         <IRanges>  <Rle>
##   [1]       17 35500001-36000000      *
##   -------
##   seqinfo: 25 sequences from an unspecified genome
```

```
segs.gr.overlap.ind <- subjectHits(hits1)[1:5]
segs.gr[segs.gr.overlap.ind]
```

```
## GRanges object with 5 ranges and 3 metadata columns:
##       seqnames            ranges strand |                 Sample Num_Probes
##          <Rle>         <IRanges>  <Rle> |            <character>  <integer>
##   [1]       17   987221-73296953      * | TCGA-3C-AAAU-10A-01D..      33859
##   [2]       17 25270517-73296953      * | TCGA-3C-AAAU-01A-11D..      24226
##   [3]       17   987221-80917016      * | TCGA-3C-AALI-10A-01D..      36977
##   [4]       17 35457542-35744709      * | TCGA-3C-AALI-01A-11D..        157
##   [5]       17 35750377-37063505      * | TCGA-3C-AALI-01A-11D..        528
##       Segment_Mean
##          <numeric>
##   [1]       0.0088
##   [2]       0.1856
##   [3]       0.0057
##   [4]       2.1456
##   [5]       1.7537
##   -------
##   seqinfo: 23 sequences from an unspecified genome; no seqlengths
```

Exercise 1:

a. Create a range for `11:69400000-69500000`.

```
# GRanges()
```

b. Find overlap between `11:69400000-69500000` and `segs.gr`.

```
# findOverlaps()
```

c. What is the `Segment_Mean` for the 2nd segment that overlaps `11:69400000-69500000`?

```
# subjectHits()
```