Using geneRxCluster

Charles C. Berry

April 24, 2017

Contents

1 Overview

The `geneRxCluster` package provides some functions for exploring genomic insertion sites originating from two different sources. Possibly, the two sources are two different gene therapy vectors. In what follows, some simulations are used to create datasets to illustrate functions in the package, but it is not necessary to follow the details of the simulations to get an understanding of the functions. More examples and details are given by Supplement 2 of Berry et al [1] available at the Bioinformatics web site.

2 Basic Use

It might be helpful to look at these help pages briefly before getting started:

Function	Purpose
critVal.target	a helper for gRxCluster
gRxCluster	the main function
gRxCluster-object	says what gRxCluster returns
gRxPlot	plots results and crtical regions
gRxSummary	quick summary of results

2.1 Reading Data from a File

The core function in the package is `gRxCluster` and it requires genomic locations and group indicators. Those basic data might be represented by a table like this:

```
chromo    pos    grp
  chr1 176812 FALSE
  chr1 191298  TRUE
  chr1 337906  TRUE
  chr1 356317  TRUE
  chr1 516904 FALSE
  chr1 661124 FALSE
    .      .      .
    .      .      .
    .      .      .
```

In R that table might be a `data.frame` or a collection of three equal length vectors. The first one here, `chromo`, indicates the chromosome. The `pos` column indicates the position on the chromosome (and note that the positions have been ordered from lowest to highest), and the `grp` vector indicates which of the two groups the row is associated with.

If a table called `exptData.txt` contained the table above, this command would read it in:

```
> df <- read.table("exptData.txt", header=TRUE)
```

2.2 Simulating Data

Here, `df` will be simulated. For a start some insertion sites are simulated according to a null distribution - i.e. the two sources are chosen according to a coin toss at each location. First the chromosome lengths are given

```
> chr.lens <- structure(c(247249719L, 242951149L, 199501827L,
+     191273063L, 180857866L, 170899992L, 158821424L, 146274826L,
+     140273252L, 135374737L, 134452384L, 132349534L, 114142980L,
+     106368585L, 100338915L, 88827254L, 78774742L, 76117153L,
+     63811651L, 62435964L, 46944323L, 49691432L, 154913754L,
+     57772954L), .Names = c("chr1", "chr2", "chr3", "chr4", "chr5",
+     "chr6", "chr7", "chr8", "chr9", "chr10", "chr11", "chr12",
+     "chr13", "chr14", "chr15", "chr16", "chr17", "chr18", "chr19",
+     "chr20", "chr21", "chr22", "chrX", "chrY"))
```

Now a sample is drawn from the chromosomes and for each chromosome a sample of positions is drawn. The function `sample.pos` is defined that samples the desired number of positions in the right range. These results are placed in a `data.frame`

```
> set.seed(13245)
> chr.names <- names(chr.lens)
> chr.factor <- factor(chr.names,chr.names)
> chrs <- sample(chr.factor,40000,repl=TRUE,
+                prob=chr.lens)
> chr.ns <- table(chrs)
> sample.pos <- function(x,y) sort(sample(y,x,repl=TRUE))
> chr.pos <-
+     mapply( sample.pos, chr.ns,chr.lens,SIMPLIFY=FALSE)
> df <-
+     data.frame(chromo=rep(chr.factor,chr.ns),
+                pos=unlist(chr.pos))
```

Now two groups are sampled as a logical vector:

```
> df$grp <-
+     rbinom(40000, 1, 0.5)==1
```

2.3 Invoking gRxCluster

With this data.frame the function can be invoked.

```
> require(geneRxCluster,quietly=TRUE)
> null.results <-
+     gRxCluster(df$chromo,df$pos,df$grp,15L:30L,nperm=100L)
> as.data.frame(null.results)[,c(-4,-5)]

  seqnames    start       end value1 value2 clump.id target.min
1     chr1 65277420 66569886      2     19        1  4.8536749
2     chr5 90199216 91786531     21      2        2  1.1589974
3     chr9 46794674 49362713     35      5        3  0.1915269
4    chr10 62347867 62907224     15      0        4  1.1043553
5    chr11 33337534 34488514      1     17        5  3.1647822
```

The function call specified window widths of 15L:30L sites and called for 100 permutations of the data with nperm=100L.

The resulting object, null.results, is a GRanges object (which is supported by the GenomicRanges package [2]) has 5 clumps. These clumps can be compared to the number of expected False Discoveries by invoking the function gRxSummary:

```
> gRxSummary( null.results )

$Clusters_Discovered
[1] 5

$FDR
```

3

```
[1] 0.6783333

$permutations
[1] 100

$targetFD
[1] 5

$call
gRxCluster(object = df$chromo, starts = df$pos, group = df$grp,
    kvals = 15L:30L, nperm = 100L, cutpt.tail.expr = critVal.target(k,
        n, target = 5, posdiff = x), cutpt.filter.expr = as.double(apply(x,
        2, median, na.rm = TRUE)))
```

The printed summary indicates 5 clusters (or clumps) were discovered, and that the estimated False Discovery Rate was 0.68 is a bit less than 1.0, which we know to be the actual False Discovery Rate. However, this is well within the bounds of variation in a simulation like this. The last part of the printout shows the values of all the arguments used in the call to gRxCluster including two that were filled in by default, and which will be discussed later on.

2.4 Simulating Clumps

Let's look at another example, but first add some true clumps to the simulation. We start by sampling chromosomes 30 times:

```
> clump.chrs <- sample(chr.factor,30,repl=TRUE,
+                   prob=chr.lens)
```

For each sample a position is chosen using the sample.pos function defined above

```
> clump.chr.pos.bound <-
+     sapply(chr.lens[clump.chrs], function(y) sample.pos(1,y))
```

For each position, the number of sites in the clump is determined:

```
> clump.site.ns <- rep(c(15,25,40),each=10)
```

For every position, nearby sites (< 1 Mbase) are sampled:

```
> clump.sites <-
+     lapply(seq_along(clump.chrs),
+             function(x) {
+                 chromo <- clump.chrs[x]
+                 n <- clump.site.ns[x]
```

4

```
+                    ctr <- clump.chr.pos.bound[x]
+                    chrLen <- chr.lens[chromo]
+                    if (ctr<chrLen/2)
+                            {
+                                 ctr + sample(1e6,n)
+                            } else {
+                                 ctr - sample(1e6,n)
+                            }
+                })
```

and grps are assigned to each clump

```
> clump.grps <- rep(0:1,15)==1
```

then a `data.frame` is constructed, added to the `df` `data.frame` and the positions are put in order:

```
> df2 <- data.frame(
+     chromo=rep(clump.chrs,clump.site.ns),
+     pos=unlist(clump.sites),
+     grp=rep(clump.grps,clump.site.ns)
+     )
> df3 <- rbind(df,df2)
> df3 <- df3[order(df3$chromo,df3$pos),]
```

Finally, the clump discovery takes place:

```
> alt.results <-
+     gRxCluster(df3$chromo,df3$pos,df3$grp,
+                15L:30L, nperm=100L)
> gRxSummary(alt.results)

$Clusters_Discovered
[1] 29

$FDR
[1] 0.1103333

$permutations
[1] 100

$targetFD
[1] 5

$call
gRxCluster(object = df3$chromo, starts = df3$pos, group = df3$grp,
    kvals = 15L:30L, nperm = 100L, cutpt.tail.expr = critVal.target(k,
        n, target = 5, posdiff = x), cutpt.filter.expr = as.double(apply(x,
        2, median, na.rm = TRUE)))
```

There were plenty of clumps discovered. Were they the simulated clumps or just False Discoveries? Several functions from the `GenomicRanges` package [2] are useful in sorting this out. Here sites in the simulated clumps are turned into a `GRanges` object.

```
> df2.GRanges <-
+     GRanges(seqnames=df2$chromo,IRanges(start=df2$pos,width=1),
+                clump=rep(1:30,clump.site.ns))
```

The function `findOverlaps` is used to map the regions in which clumps were found to the sites composing those simulated clumps, then the function `subjectHits` indicates which of the simulated clumps were found.

```
> clumps.found <- subjectHits(findOverlaps(alt.results,df2.GRanges))
```

Finally, the number of sites in the simulated clumps that are covered by each estimated clump is printed.

```
> matrix(
+     table(factor(df2.GRanges$clump[ clumps.found ],1:30)),
+     nrow=10,dimnames=list(clump=NULL,site.ns=c(15,25,40)))
```

```
       site.ns
clump    15 25 40
   [1,]   9 25 32
   [2,]   0 21 19
   [3,]  15 23 40
   [4,]   0 21 36
   [5,]   0 18 40
   [6,]   0 25 29
   [7,]   0 17 40
   [8,]  11 25 40
   [9,]  15 22 40
  [10,]   0 23 31
```

Notice that fewer than half of the clumps consisting of just 15 sites are found, the clumps of 25 sites are usually found, but usually all of the sites composing each clump are not found. The clumps formed from 40 sites are found and all or almost all of the sites in each clump are found.

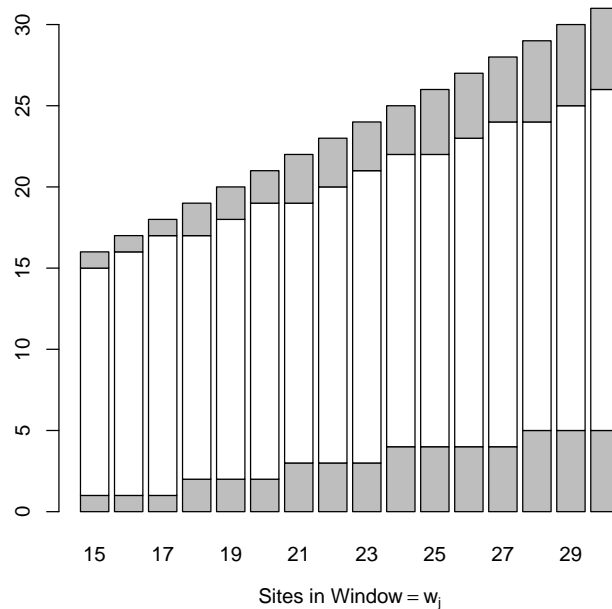And here the clumps that are False Discoveries are counted by using the `countOverlaps` function

```
> sum( countOverlaps(alt.results, df2.GRanges ) == 0 )
```

```
[1] 5
```

3 Customizing Critical Regions and Filters

The critical regions used above can be displayed like this:

```
> gRxPlot(alt.results,method="criticalRegions")
```


Sites in Window = w_j

Notice that the regions are not perfectly symmetrical. This is because the proportions of the two classes are not exactly equal:

```
> xtabs(~grp, df3)

grp
FALSE  TRUE
20278 20522
```

The `gRxCluster` function provides a means of using another set of criitical regions and another filter expression. The expression for settings up critical regions is found in the is found in the `metadata()` slot of `alt.results` in the `$call` component:

```
> as.list(metadata(alt.results)$call)[['cutpt.tail.expr']]

critVal.target(k, n, target = 5, posdiff = x)
```

7

The expression is evaluated in an enviroment that has objects `k`, `n`, and an object called `x` that the expression may use. The object `k` is a set of values for the number of sites to include in a window, `n` is the results of `table(df3$grp)`, and `x` is a matrix of the lagged differences of `df3[,"pos"]`. The lags of order `(15:30)-1` (setting those that cross chromosome boundaries to `NA`) make up the columns of `x`.

One obvious change that a user might make is to reset the value of `target`.

```
> generous.target.expr <-
+     quote(critVal.target(k,n, target=20, posdiff=x))
> generous.results <-
+     gRxCluster(df3$chromo,df3$pos,df3$grp,
+                15L:30L,nperm=100L,
+                cutpt.tail.expr=generous.target.expr)
> gRxSummary(generous.results)

$Clusters_Discovered
[1] 43

$FDR
[1] 0.3079545

$permutations
[1] 100

$targetFD
[1] 20

$call
gRxCluster(object = df3$chromo, starts = df3$pos, group = df3$grp,
    kvals = 15L:30L, nperm = 100L, cutpt.tail.expr = generous.target.expr,
    cutpt.filter.expr = as.double(apply(x, 2, median, na.rm = TRUE)))
```

Many more discoveries are made, but look at the count of false discoveries:

```
> sum( 0==countOverlaps(generous.results,df2.GRanges))

[1] 18
```

The filter function is also found in the `metadata()` slot of `alt.results` in the `$call` component:

```
> as.list(metadata(alt.results)$call)[['cutpt.filter.expr']]

as.double(apply(x, 2, median, na.rm = TRUE))
```

`alt.result` filtered out the windows whose widths were less than the median number of bases. The expression is evaluated in the enviroment as before, but

8

only the object x has been added in at the time the expression is called. If filtering is not desired it can be tunred off by using an expression that returns values higher than any seen in x such as this:

```
> no.filter.expr <- quote(rep(Inf,ncol(x)))
> no.filter.results <-
+     gRxCluster(df3$chromo,df3$pos,df3$grp,15L:30L,nperm=100L,
+                cutpt.filter.expr=no.filter.expr)
> gRxSummary(no.filter.results)

$Clusters_Discovered
[1] 26

$FDR
[1] 0.1237037

$permutations
[1] 100

$targetFD
[1] 5

$call
gRxCluster(object = df3$chromo, starts = df3$pos, group = df3$grp,
    kvals = 15L:30L, nperm = 100L, cutpt.filter.expr = no.filter.expr,
    cutpt.tail.expr = critVal.target(k, n, target = 5, posdiff = x))
```

The effect of using non-specific filters to increase power is applied in gene-expression microarray studies [3]. The less stringent filtering results in fewer discoveries, but the number of false discoveries also decreased:

```
> sum( 0==countOverlaps(no.filter.results,df2.GRanges))

[1] 4
```

Here a more stringent filter is used

```
> hard.filter.expr <-
+     quote(apply(x,2,quantile, 0.15, na.rm=TRUE))
> hard.filter.results <-
+     gRxCluster(df3$chromo,df3$pos,df3$grp,15L:30L,
+                nperm=100L,
+                cutpt.filter.expr=hard.filter.expr)
> gRxSummary(hard.filter.results)

$Clusters_Discovered
[1] 27
```

```
$FDR
[1] 0.1507143

$permutations
[1] 100

$targetFD
[1] 5

$call
gRxCluster(object = df3$chromo, starts = df3$pos, group = df3$grp,
    kvals = 15L:30L, nperm = 100L, cutpt.filter.expr = hard.filter.expr,
    cutpt.tail.expr = critVal.target(k, n, target = 5, posdiff = x))
```

The number of discoveries here needs to be corrected for the number of false discoveries if comparisons are to be made:

```
> sum( 0==countOverlaps(hard.filter.results,df2.GRanges))

[1] 3
```

It seems to do a bit better than the other two alternatives when true and false discovery numbers are considered.

References

[1] Charles C. Berry and Karen E. Ocwieja and Nirvav Malani and Frederic D. Bushman. Comparing DNA site clusters with Scan Statistics. *Bioinformatics*, doi: 10.1093/bioinformatics/btu035, 2014.

[2] 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):e1003118, 2013.

[3] R. Bourgon, R. Gentleman, and W. Huber. Independent filtering increases detection power for high-throughput experiments. *Proceedings of the National Academy of Sciences*, 107(21):9546, 2010.