CNV Analysis

Isaac De la Hoz

13 de mayo de 2019

Copy number variants detection using *exomeCopy* R package

exomeCopy is an R package implementing a hidden Markov model for predicting copy number variants (CNVs) from exome sequencing experiments without paired control experiments as in tumor/normal sequencing. It models read counts in genomic ranges using negative binomial emission distributions depending on a hidden state of the copy number and on positional covariates such as GC-content and background read depth.Normalization and segmentation are performed simultaneously, eliminating the need for preprocessing of the raw read counts.

Methodology

Importing experiment data

Due to we are working with exome data, we have to provide an annotation file defining the exon regions. This information is needed to perform the later analysis. The data was obtained from UCSC webpage. It was post-processed through the following code in order to remove non-relevant information and sort the records.

```
cat target_exons.bed | grep -v "chrX[[:space:]]" | grep -v "chrY[[:space:]]" | \
                sort -k 1V,1 -k 2n,2 -k 3n,3 | uniq > AutosomalExons.bed
```

Once we obtained the annotation file, all directories were specified to import the experiment data. All this data were stored as Granges objects.

```
library(exomeCopy)
library(GenomicRanges)
setwd("~/data/WES_obesity/CNV")
target.file <- "~/data/WES_obesity/AutosomalExons.bed"
#Sample 12 and 15 are nor included into the anaysis bucause of their bad quality
bam.files <- list.files(path="~/data/WES_obesity/Data",
                        pattern = ".bam$", full.names = TRUE)[-c(12,15)]
sample.names <- gsub(".bam$", "", (list.files(path="~/data/WES_obesity/Data",
                                              pattern = ".bam$",
                                              full.names = FALSE)))[-c(12,15)]
reference.file <- "~/data/WES_obesity/hg38.fa"
target.df <- read.delim(target.file, header = FALSE)
#The target file is converted in a Grange object
target <- GRanges(seqname = target.df[, 1],
                  IRanges(start = target.df[,2] + 1, end = target.df[, 3]))
target <- sortSeqlevels(target)
```

Counting reads in genomic ranges

The next step consist in counting reads from the BAM files in genomic ranges covering the targeted regions. The function `countBamInRanges` returns a vector of counts, representing the number of sequenced read starts (leftmost position regardless of strand) with mapping quality above a minimum threshold for each genomic range. The following loop was created in order to count all bam files iteratively.

```
counts <- target
for (i in 1:length(bam.files)){
  print(sprintf("Processing %s", sample.names[i]))
  mcols(counts)[[sample.names[i]]]<-countBamInGRanges(bam.files[i],target)
}
```

Calculating GC-content and background depth

In order to perform the function `exomeCopy` it is necessary to calculate the GC-content and the background depth. The function `getGContent` was used to calculate it.

```
counts$GC <- getGCcontent(target, reference.file)
```

Once the GC-content was calculated, the background read depth was generated through a simple function which performs 3 simple steps:

- Given a vector of names of samples to be used as background, extract the read counts data frame from the GRanges object.
- It divide each sample by its mean read count (column means).
- It calculate the median of these normalized read counts (row medians)

The relationship between read counts and GC-content over the ranges varies across protocols and samples. It can be roughly approximated per sample using second-order polynomial terms of GC-content. We store the square of GC-content as a new value column. Other functions of GC-content could be used as well. We also store the width of the ranges as a value column.

```
counts$GC.sq <- counts$GC^2
counts$bg <- generateBackground(sample.names, counts, median)counts$GC.sq <- counts$GC^2
counts$bg <- generateBackground(sample.names, counts, median)
## All zero background coverage records are removed in order to avoid further problems
counts2 <- counts[counts$bg >0,]
counts2$log.bg <- log(counts$bg + 0.1)
counts2$width <- width(counts2)
```

Running `exomeCopy` model

Once we had calculated all parameters before explained, we were able to run the exome copy function. This function is designed to process one chromosome of one sample each time. Considering that, the following wrapper function was created in order to loop `exomeCopy` over all samples and chromosomes.

```
runExomeCopy <- function(sample.name, seqs) {
  lapply(seqs, function(seq.name) exomeCopy(counts2[seqnames(counts2) == seq.name],
                                            sample.name, X.names =
                                              c("log.bg","GC", "GC.sq", "width"), S = 0:4, d = 2))
}
seqs<-seqlevels(counts2)
fit.list <- lapply(sample.names, runExomeCopy, seqs)
```

After fitting, we call the function `compileCopyCountSegments` on the ExomeCopy object, which provides the segmentation with the predicted copy number, the log odds of read counts being emitted from predicted copy count over normal copy count, the number of input genomic ranges contained within each segment, the number of targeted basepairs contained in the segments, and the name of the sample to help compile the segments across samples.

```
compiled.segments <- compileCopyCountSegments(fit.list)
#The result Grange object looks life this:
```

```r
head(compiled.segments)
```

```
## GRanges object with 6 ranges and 5 metadata columns:
##       seqnames        ranges strand | copy.count  log.odds   nranges
##          <Rle>     <IRanges>  <Rle> |  <integer> <numeric> <numeric>
##   [1]     chr1   12975-17368      * |          2         0         6
##   [2]     chr1   17369-70005      * |          4     10.77         6
##   [3]     chr1 131025-187577      * |          2         0        11
##   [4]     chr1 188130-925800      * |          0     28.24        63
##   [5]     chr1 925741-925800      * |          2         0         1
##   [6]     chr1 925922-947060      * |          4      8.36        62
##       targeted.bp sample.name
##         <integer> <character>
##   [1]        <NA>      141276
##   [2]        <NA>      141276
##   [3]        <NA>      141276
##   [4]        <NA>      141276
##   [5]        <NA>      141276
##   [6]        <NA>      141276
##   -------
##   seqinfo: 22 sequences from an unspecified genome; no seqlengths
```

Overlap analysis of CNVs with functional genomic regions

From the output obtained from exomeCopy we could distinguish which region was under or over-represented according to the copy count number. The annotation with a copy count number different to 2 (the normal count number for diploid organism) were considering as over represented (copy count higher than 2) or under represented (copy count lower than 2). Considering this premise, we selected all annotation with a copy count different than 2 and we discarded all non-useful information for the later analysis.

```r
finalCNV<- as.data.frame(compiled.segments[c("sample.name", "copy.count")])
names(finalCNV)[7]<-"state"
finalCNV<-finalCNV[,c(1,2,3,6,7)]
finalCNV<-finalCNV[finalCNV$state!=2,]
#We group the calls by sample ID, resulting in a GRangesList.
grl <- makeGRangesListFromDataFrame(finalCNV,
                                    split.field="sample.name", keep.extra.columns=TRUE)
#The Grangelist looks like this:
```

```r
grl
```

```
## GRangesList object of length 15:
## $141276
## GRanges object with 11063 ranges and 1 metadata column:
##           seqnames            ranges strand |     state
##              <Rle>         <IRanges>  <Rle> | <integer>
##       [1]     chr1       17369-70005      * |         4
##       [2]     chr1     188130-925800      * |         0
##       [3]     chr1     925922-947060      * |         4
##       [4]     chr1     970277-975108      * |         4
##       [5]     chr1     974573-998051      * |         0
##       ...      ...               ...    ... .       ...
##   [11059]    chr22 50577775-50578659      * |         0
##   [11060]    chr22 50580565-50623326      * |         1
##   [11061]    chr22 50678585-50777981      * |         1
##   [11062]    chr22 50777952-50780718      * |         0
##   [11063]    chr22 50783039-50801309      * |         0
##
## ...
## <14 more elements>
## -------
## seqinfo: 22 sequences from an unspecified genome; no seqlengths
```

Summarizing individual CNV calls across a population

In CNV analysis, it is often of interest to summarize individual calls across the population, (i.e. to define CNV regions), for subsequent association analysis with expression and phenotype data. In the simplest case, this just merges overlapping individual calls into summarized regions. In this case, we used the approach from CNVRuler to summarize CNV calls to CNV regions. This trims low-density areas as defined by the density argument, which is set here to <10% of the number of calls within a summarized region.

```
##CNVRANger. CNV Enrichment
library(CNVRanger)
grlCNV<-sort(grl)
cnvrs <- populationRanges(grlCNV, density=0.1)
cnvrs
```

```
## GRanges object with 1457 ranges and 2 metadata columns:
##          seqnames              ranges strand |      freq        type
##             <Rle>           <IRanges>  <Rle> | <numeric> <character>
##      [1]     chr2     8860695-8861142      * |         4        both
##      [2]     chr2     8862271-8862722      * |         5        both
##      [3]     chr2     8864170-8864234      * |         4        both
##      [4]     chr2     8868446-8868549      * |         4        both
##      [5]     chr2     8873108-8873300      * |         4        both
##      ...      ...                 ...    ... .       ...         ...
##   [1453]    chr12 133238455-133238549      * |         2        loss
##   [1454]    chr13   20403666-20406128      * |         6        both
##   [1455]    chr13   20432102-20567785      * |        13        both
##   [1456]    chr17       118383-118578      * |         1        loss
##   [1457]    chr19         70652-70976      * |         3        loss
##   -------
##   seqinfo: 22 sequences from an unspecified genome; no seqlengths
```

Once individual CNV calls have been summarized across the population, it is typically of interest whether the resulting CNV regions overlap with functional genomic regions such as genes, promoters, or enhancers. As a certain amount of overlap can be expected just by chance, an assessment of statistical significance is needed to decide whether the observed overlap is greater (enrichment) or less (depletion) than expected by chance.

The regioneR package implements a general framework for testing overlaps of genomic regions based on permutation sampling. This allows to repeatedly sample random regions from the genome, matching size and chromosomal distribution of the region set under study (here: the CNV regions). By recomputing the overlap with the functional features in each permutation, statistical significance of the observed overlap can be assessed.

In the following code we extracted the annotations of protein coding genes from AnnotationHub in order to prove if the tool works properly with our data.

```
library(AnnotationHub)
ah <- AnnotationHub()
ahDb <- query(ah, pattern = c("Homo Sapiens", "EnsDb"))
```

```r
ahEdb <- ahDb[["AH69187"]]
#Interesting regions
##Genes
hg.genes <- genes(ahEdb)
sel.genes <- hg.genes[hg.genes$gene_biotype == "protein_coding"]
seqlevelsStyle(sel.genes)<-"UCSC"
```

Once we had the annotations, we perform the test with 10000 permutations (ntimes=10000), while maintaining chromosomal distribution of the CNV region set (per.chromosome=TRUE). Furthermore, we use the option count.once=TRUE to count an overlapping CNV region only once, even if it overlaps with 2 or more genes. We also allow random regions to be sampled from the entire genome (mask=NA).

```r
library(regioneR)
library(BSgenome.Hsapiens.UCSC.hg38.masked)
res <- suppressWarnings(overlapPermTest(A=cnvrs, B=sel.genes, ntimes=1000,
                                        genome="BSgenome.Hsapiens.UCSC.hg38.masked",
                                        mask=NA, per.chromosome=TRUE, count.once=TRUE))
res
```

```
## $numOverlaps
## P-value: 0.000999000999000999
## Z-score: 33.3608
## Number of iterations: 1000
## Alternative: greater
## Evaluation of the original region set: 1386
## Evaluation function: numOverlaps
## Randomization function: randomizeRegions
##
## attr(,"class")
## [1] "permTestResultsList"
```