VanillaICE: Hidden Markov Models for the Assessment of Chromosomal Alterations using High-throughput SNP Arrays

Robert Scharpf

April 27, 2020

Abstract

This package provides an implementation of a hidden Markov Model to identify copy number alterations from high throughput SNP arrays.

1 From processed SNP summaries

The starting point for this section of the vignette are B allele frequencies and log R ratios that are available from software such as GenomeStudio and the R package *crlmm*. In this section, we assume that the low-level summaries are available in a plain text file – one file per sample. For users of the *crlmm* package for preprocessing, please refer to the *crlmmDownstream* vignette.

To illustrate the workflow for copy number analysis, this package provides Illumina 610k array data for one Hapmap trio and one oral cleft trio preprocessed by GenomeStudio [1]. To keep the size of this package small, the GenomeStudio-processed files for these 6 samples have been parsed to include only columns relevant to copy number analyses and $\approx$ 11k markers that are spanned by or border putative CNVs. The code chunks provided below would be the same if the full files were included.

```
> library(VanillaICE)
> library(foreach)
> registerDoSEQ()
> extdir <- system.file("extdata", package="VanillaICE", mustWork=TRUE)
> files <- list.files(extdir, pattern="FinalReport")
```

The first several lines of these files have the following information:

```
[Header]
BSGT Version,3.3.7
Processing Date,5/2/2009 12:08 AM
Content,,Human610-Quadv1_B.bpm
Num SNPs,620901
Total SNPs,620901
Num Samples,7599
Total Samples,7599
File ,1664 of 7599
[Data]
SNP Name,Allele1 - AB,Allele2 - AB,B Allele Freq,Log R Ratio
cnvi0005126,-,-,0.0085,-0.7214
cnvi0005127,-,-,0.0081,-0.8589
cnvi0005128,-,-,0.0063,-0.164
```

The physical annotation of the markers in the genome can be downloaded from on-line resources or from annotation packages available from Bioconductor. Here, we provide a plain-text file containing the genomic annotation for the markers based on build UCSC build hg18:

```
> list.files(extdir, pattern="SNP_info")
```

```
[1] "SNP_info.csv"
```

The annotation file contains the the name of the SNP or nonpolymorphic marker ('Name'), an indicator for whether the marker is a SNP ('Intensity Only'), the chromosome name ('Chr'), and the genomic physical position ('Position'). The columns for the SNP annotation can be in any order and have different variable labels than the file provided with this package. Below, we copy the header of the provided annotation file:

```
Chr,Position,Name,Intensity Only
11,45380778,cnvi0005126,1
11,45382079,cnvi0005127,1
11,46561032,cnvi0005128,1
```

1.1 Organizing marker annotation

We require that marker-level annotation is represented as a `GRanges`-derived class. To read the plain text annotation file, we use the `fread` provided in the *data.table* package. In addition, we define an indicator for whether the marker is polymophic using the 'Intensity Only' flag. The `SnpGRanges` class created from the `fgr` object in the following code-chunk ensures that this binary indicator is created and can be reliably accessed in downstream processing.

```
> require(data.table)
> features <- suppressWarnings(fread(file.path(extdir, "SNP_info.csv")))
> fgr <- GRanges(paste0("chr", features$Chr), IRanges(features$Position, width=1),
+                isSnp=features[["Intensity Only"]]==0)
> fgr <- SnpGRanges(fgr)
> names(fgr) <- features[["Name"]]
```

Ideally, one should also include the genome build and information on the chromosome lengths appropriate to the build. Here, we extract the metadata on the chromosomes using BSgenome Bioconductor package for the hg18 build. Finally, we sort the `fgr` object such that the chromosomes are ordered by their `seqlevels` and the markers are ordered by their genomic position along the chromosome.

```
> library(BSgenome.Hsapiens.UCSC.hg18)
> sl <- seqlevels(BSgenome.Hsapiens.UCSC.hg18)
> seqlevels(fgr) <- sl[sl %in% seqlevels(fgr)]
> seqinfo(fgr) <- seqinfo(BSgenome.Hsapiens.UCSC.hg18)[seqlevels(fgr),]
> fgr <- sort(fgr)
```

1.2 Organizing the marker-level summaries

The abbreviated plain text files included with this package and containing log R ratios and B allele frequencies from 2 trios are listed below.

```
> files <- list.files(extdir, full.names=TRUE, recursive=TRUE, pattern="FinalReport")
```

We will parse these files into a specific format such that all downstream steps for copy number estimation no longer depend on the specific format of the source files. At this point, we can encapsulate the the names of the source files ('sourcePaths'), the location of where we intend to store the parsed data ('parsedPath'), and the genomic marker annnotation created in the preceding section in a single object called an `ArrayViews`.

```
> ##
> ## A directory to keep parsed files
> ##
> parsedDir <- tempdir()
> views <- ArrayViews(rowRanges=fgr, sourcePaths=files,
+                     parsedPath=parsedDir)
> lrrFile(views) <- file.path(parsedDir, basename(fileName(views, "lrr")))
> views@bafFiles <- file.path(parsedDir, basename(fileName(views, "baf")))
> views@gtFiles <- file.path(parsedDir, basename(fileName(views, "gt")))
> colnames(views) <- gsub(".csv", "", colnames(views))
> show(views)

class 'ArrayViews'
   No. files  : 6
   No. markers: 11780
```

Because the format of the source files depends on upstream software and version number within software, our approach to parsing these files is to read one file and to store the appropriate metadata from this file so that subsequent files can be parsed in a similar fashion. We use the `fread` to read in the first file.

```
> ## read the first file
> dat <- fread(files[1], skip="[Data]")
> head(dat,n=3)

      SNP Name Allele1 - AB Allele2 - AB B Allele Freq Log R Ratio
1: cnvi0005126            -            -        0.0085     -0.7214
2: cnvi0005127            -            -        0.0081     -0.8589
3: cnvi0005128            -            -        0.0063     -0.1640
```

Next, we select which columns we plan to keep. Again, the required data for downstream processing is the name of the SNP identifier, the log R ratios, and B allele frequencies.

```
> ## information to store on the markers
> select_columns <- match(c("SNP Name", "Allele1 - AB", "Allele2 - AB",
+                           "Log R Ratio", "B Allele Freq"), names(dat))
```

We also specify the order in which we will store the marker-level summaries by matching the `rownames` of the `views` object with the names of the markers in the source file:

```
> index_genome <- match(names(fgr), dat[["SNP Name"]])
```

Similar to the parameter classes defined in *Rsamtools*, we encapsulate the information for parsing the columns and rows of the source files in a class. In addition, we specify which variable names in the source file refers to log R ratios ('cnvar'), B allele frequences ('bafvar'), and genotypes ('gtvar').

```
> scan_params <- CopyNumScanParams(index_genome=index_genome,
+                                  select=select_columns,
+                                  cnvar="Log R Ratio",
+                                  bafvar="B Allele Freq",
+                                  gtvar=c("Allele1 - AB", "Allele2 - AB"))
```

The `parseSourceFile` will parse a single file in the `views` object (by default, the first file) according to the parameters for reading the data in the `scan_params` object and write the output to the `parsedPath` directory. In particular, the `parseSourceFile` returns `NULL`.

```
> parseSourceFile(views, scan_params)
```

Apart from confirming their existance, the user should not have a need to directly access the parsed files. Utilities for querying these files are provided through the `views` object.

```
> head(list.files(parsedPath(views)), n=3)
```

```
[1] "FinalReport1664_baf.rds" "FinalReport1664_gt.rds"
[3] "FinalReport1664_lrr.rds"
```

1.3 Accessors for the parsed data

The preference for writing the parsed data to disk rather than keeping the data in RAM is simply that the latter does not scale to projects involving thousands of samples. For the former, slices of the parsed data easily be accessed from the `parsedPath` directory via methods defined for the `ArrayViews` class. For example, one can use accessors for the low-level summaries directly: `lrr`, `baf`, and `genotypes` for log R ratios, B allele frequencies, and genotypes, respectively. The user has the option of either subsetting the views object or subsetting the matrix returned by the accessor to extract the appropriate data slice. In the following examples, we access data on the first 2 markers.

```
> lrr(views)[1:2, ]
```

```
           FinalReport1664.txt FinalReport1675.txt
rs12789205               0.271              -0.169
rs2114088                0.114              -0.182
           FinalReport1686.txt FinalReport6841.txt
rs12789205               0.191              -0.084
rs2114088               -0.114               0.253
           FinalReport6872.txt FinalReport6903.txt
rs12789205               0.067              -0.101
rs2114088               -0.076              -0.043
```

```
> ## or
> lrr(views[1:2, ])
```

```
           FinalReport1664.txt FinalReport1675.txt
rs12789205               0.271              -0.169
rs2114088                0.114              -0.182
           FinalReport1686.txt FinalReport6841.txt
rs12789205               0.191              -0.084
rs2114088               -0.114               0.253
           FinalReport6872.txt FinalReport6903.txt
rs12789205               0.067              -0.101
rs2114088               -0.076              -0.043
```

```
> ## B allele frequencies
> baf(views[1:2, ])
```

```
           FinalReport1664.txt FinalReport1675.txt
rs12789205               0.999               0.529
rs2114088                0.000               0.000
           FinalReport1686.txt FinalReport6841.txt
rs12789205               0.549               0.481
rs2114088                0.000               0.000
           FinalReport6872.txt FinalReport6903.txt
rs12789205               0.482               0.488
rs2114088                0.000               0.493
```

```
> ## Use :: to avoid  masking by function of the same name in crlmm
> VanillaICE::genotypes(views)[1:2, ]
```

```
           FinalReport1664.txt FinalReport1675.txt
rs12789205                   3                   2
rs2114088                    1                   1
           FinalReport1686.txt FinalReport6841.txt
rs12789205                   2                   2
rs2114088                    1                   1
           FinalReport6872.txt FinalReport6903.txt
rs12789205                   2                   2
rs2114088                    1                   2
```

More often, it is useful to extract the low-level data in a `RangedSummarizedExperiment`-derived object such that meta-data on the samples remains bound to the columns of the assay data (log R ratios / B allele frequencies) and meta-data on the rows remains bound to the rows of the assay data. This is accomplished by applying the `SnpExperiment` function to a `views` object. In the following example, we create a `RangedSummarizedExperiment`-derived object for the first three samples in the `views` object.

```
> snp_exp <- SnpExperiment(views[, 4:5])
> show(snp_exp)
```

```
class: SnpArrayExperiment
dim: 11780 2
metadata(0):
assays(2): cn baf
rownames(11780): rs12789205 rs2114088 ... rs5994329 rs1055232
rowData names(1): isSnp
colnames(2): FinalReport6841.txt FinalReport6872.txt
colData names(0):
```

2 Hidden Markov model

2.1 HMM parameters

The hidden Markov model for inference of germline copy number from SNP array data was originally described in Scharpf *et al.* [3] but has evolved to focus more on B allele frequencies than genotype call probabilities [the latter requires a dependency on the *crlmm* package]. Many of the the updates for computing emission probabilities have been described more recently [2].

Fitting the hidden Markov model to the marker-level summaries requires extracting the relevant data from a `views` object using the `SnpExperiment` function illustrated in the previous section. All user-level parameters relevant for fitting the HMM are specified in a parameter object for the emission probabilities and a parameter object for the transition probabilities. In the following code-chunk, we create a parameter object for the emission probabilities.

```
> param <- EmissionParam()
```

The easiest way to specify or change parameters for computing emission probabilities is through the `EmissionParam` constructor. For example, here we create a parameter object with the `temper` parameter set to 1/2:

```
> param <- EmissionParam(temper=0.5)
> show(param)
```

```
EmissionParam :
CN mean:  -2, -0.4, 0, 0, 0.4, 1
CN sd:  0.6, 0.3, 0.3, 0.3, 0.3, 0.3
BAF mean:  0, 0.25, 0.33, 0.5, 0.67, 0.75, 1
BAF sd:  0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1
max # of EM updates:  5
initial state probabilities:  0.17, 0.17, 0.17, 0.17, 0.17, 0.17
tempering scalar:  0.5
model homozygous regions:  FALSE
See temper(), cn_means(), cn_sds(),...
```

Similarly, the constructor for the `TransitionParam` class can be used to specify alternatives to the defaults. By not specifying an `TransitionParam` object, default values created by the constructor are used by the HMM in the following example.

2.2 Fitting the HMM

The function `hmm2` fits the 6-state HMM to the `snp_exp` object, returning a list-derived class (a `HMMList`) of the same length as the number of samples.

```
> fit <- hmm2(snp_exp, param)
> show(fit)
```

```
An object of class 'HMMList'
  Length:  2
  Use [[j]] to retrieve results for the jth sample.
  Use unlist to coerce to a 'GRanges' object.
```

```
> length(fit)
```

```
[1] 2
```

```
> ## HMM object for the first sample
> fit[[1]]
```

```
Object of class 'HMM'
  granges (no. segments)  : 7
  no. duplications        : 2
  no. hemizygous deletions: 0
  no. homozygous deletions: 0
See posterior(), segs(), cnvSegs(), getHmmParams(),
```

3 Inspecting, Filtering, and plotting HMM results

Several methods are defined for the `HMMList` class to facilitate the selection of candidate CNVs, the filtering of technical artifacts, and to enable visualization of the HMM calls in the context of the marker-level summaries.

The `fit` object is an instance of the `HMMList` object of length three, each element corresponding to a sample:

```
> names(fit)

[1] "FinalReport6841.txt" "FinalReport6872.txt"
```

The R method \\ can be used to access the HMM summary for the results for the jth sample. For example, here we extract the HMM summary for the 2nd sample:

```
> show(fit[[2]])

Object of class 'HMM'
  granges (no. segments)  : 13
  no. duplications        : 1
  no. hemizygous deletions: 4
  no. homozygous deletions: 0
See posterior(), segs(), cnvSegs(), getHmmParams(),
```

Alternatively, we can use `unlist` to create a single `GRanges`-derived class containing the segmentation of all three samples

```
> head(unlist(fit), n=3)

HmmGRanges object with 3 ranges and 4 metadata columns:
      seqnames            ranges strand | numberFeatures     state
         <Rle>         <IRanges>  <Rle> |      <integer> <integer>
  [1]    chr11 43132366-55116789      * |           1445         3
  [2]    chr11 55124465-55204003      * |             29         5
  [3]    chr11 55209499-67164817      * |           1963         3
         prCall                  id
      <numeric>         <character>
  [1]         1 FinalReport6841.txt
  [2]         1 FinalReport6841.txt
  [3]         1 FinalReport6841.txt
  -------
  seqinfo: 3 sequences from hg18 genome
```

or a `GRangesList` by splitting on the sample id:

```
> grl <- split(unlist(fit), unlist(fit)$id)
```

3.1 Filters

There are several meta-data columns stored in the `GRanges`-derived summary useful for filtering the set of genomic intervals. All the available parameters for filtering the `fit` object are stored in the parameter class `FilterParam` of which an instance can be created by its constructor of the same name without any arguments.

```
> filter_param <- FilterParam()
> show(filter_param)

An object of class 'FilterParam'
   min. posterior probability of CNV call: 0.99
   min. no. of markers spanned by segment: 10
   min. width of segment                 : 1
   selected HMM states                   : 1, 2, 3, 4, 5, 6
   selected seqnames                     : chr1, chr2, chr3, chr4, chr5, chr6 ...
```

To apply the default filter parameters to the `fit` object, we use the `cnvFilter` function. The `cnvFilter` function returns only the set of genomic ranges satisfying the filters. For example,

```
> cnvFilter(fit, filter_param)

HmmGRanges object with 18 ranges and 4 metadata columns:
       seqnames            ranges strand | numberFeatures     state
          <Rle>         <IRanges>  <Rle> |      <integer> <integer>
   [1]    chr11 43132366-55116789      * |           1445         3
   [2]    chr11 55124465-55204003      * |             29         5
   [3]    chr11 55209499-67164817      * |           1963         3
   [4]    chr12    21054-17482570      * |           4591         3
   [5]    chr22 14456412-23994408      * |           2244         3
   ...      ...               ...    ... .            ...       ...
  [14]    chr22 18710895-19048116      * |             28         3
  [15]    chr22 19066315-19819918      * |            211         2
  [16]    chr22 19842333-23983992      * |           1099         3
  [17]    chr22 23991725-24236803      * |             63         5
  [18]    chr22 24237905-29381906      * |           1441         3
          prCall                  id
       <numeric>         <character>
   [1]         1 FinalReport6841.txt
   [2]         1 FinalReport6841.txt
   [3]         1 FinalReport6841.txt
   [4]         1 FinalReport6841.txt
   [5]         1 FinalReport6841.txt
   ...       ...                 ...
  [14]         1 FinalReport6872.txt
  [15]         1 FinalReport6872.txt
  [16]         1 FinalReport6872.txt
  [17]         1 FinalReport6872.txt
  [18]         1 FinalReport6872.txt
  -------
  seqinfo: 3 sequences from hg18 genome
```

To select only the segments with altered copy number states (states 1, 2, 5, and 6) on chromosome 22, we can define the filter parameters as follows:

```
> select_cnv <- FilterParam(state=c("1", "2", "5", "6"), seqnames="chr22")
> cnvs <- cnvFilter(fit, select_cnv)
> cnvs
```

```
HmmGRanges object with 6 ranges and 4 metadata columns:
      seqnames            ranges strand | numberFeatures     state
         <Rle>         <IRanges>  <Rle> |      <integer> <integer>
  [1]    chr22 23999142-24240667      * |             65         5
  [2]    chr22 17202486-17388108      * |             44         2
  [3]    chr22 17405381-17633332      * |             48         2
  [4]    chr22 17690812-18693299      * |            281         2
  [5]    chr22 19066315-19819918      * |            211         2
  [6]    chr22 23991725-24236803      * |             63         5
         prCall                  id
      <numeric>         <character>
  [1]         1 FinalReport6841.txt
  [2]         1 FinalReport6872.txt
  [3]         1 FinalReport6872.txt
  [4]         1 FinalReport6872.txt
  [5]         1 FinalReport6872.txt
  [6]         1 FinalReport6872.txt
  -------
  seqinfo: 3 sequences from hg18 genome
```

3.2 Visualization

3.2.1 Trellis graphics for low-level summaries

Visualization of the CNVs in the context of the lower-level summaries is acheived through a combination of grid and the grid-derived graphics provided by *lattice*. The `xyplotList` and constructor `HmmTrellisParam` should be sufficient for producing a decent graphic using the default settings. In the following code chunk, we create a `trellis` object for each CNV segment.

```
> trellis_param <- HmmTrellisParam()
> cnvList <- split(cnvs, cnvs$id)
> figList <- xyplotList(cnvList, snp_exp, trellis_param)
> names(figList)
```

```
[1] "FinalReport6841.txt" "FinalReport6872.txt"
```

Each element in `figList` is a `trellis` object and can be displayed using `print`.

```
> class(figList[["FinalReport6841.csv"]][[1]])
```

```
[1] "NULL"
```

3.2.2 Layout using grid

```
> fig1 <- figList[["FinalReport6841.csv"]][[1]]
```

```
> vps <- viewports()
> xygrid(fig1, vps, cnvList[[1]][1])
```

```
NULL
```

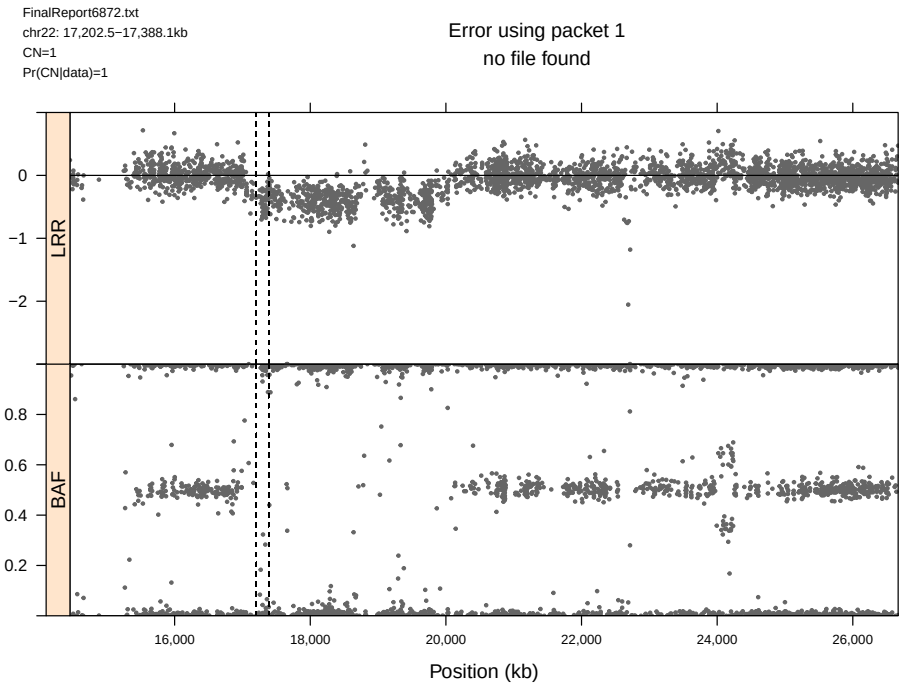
A hemizygous deletion on chromosome 22 is split into too many regions.

```
> cnvs_sample2 <- cnvList[[2]]
> cnvs_sample2
```

Figure 1: A single-copy duplication (state 5).

```
HmmGRanges object with 5 ranges and 4 metadata columns:
      seqnames            ranges strand | numberFeatures     state
         <Rle>         <IRanges>  <Rle> |      <integer> <integer>
  [1]    chr22 17202486-17388108      * |             44         2
  [2]    chr22 17405381-17633332      * |             48         2
  [3]    chr22 17690812-18693299      * |            281         2
  [4]    chr22 19066315-19819918      * |            211         2
  [5]    chr22 23991725-24236803      * |             63         5
         prCall                  id
      <numeric>         <character>
  [1]         1 FinalReport6872.txt
  [2]         1 FinalReport6872.txt
  [3]         1 FinalReport6872.txt
  [4]         1 FinalReport6872.txt
  [5]         1 FinalReport6872.txt
  -------
  seqinfo: 3 sequences from hg18 genome
```

```
> xygrid(figList[[2]][[1]], vps, cnvs_sample2[1])
```



Combining the adjacent hemizygous deletions through `reduce` provides a more satisfying segmentation of this data.

```
> cnvs_sample2r <- reduce(cnvs_sample2, min.gapwidth=500e3)
> fig2 <- xyplotList(cnvs_sample2r, snp_exp)
```

Parallelization

As the HMM is fit independently to each sample, parallelization is straightforward. In the following unevaluated code chunk, we set up a parallel environment using the R packages *snow* and *foreach*.

```
> library(foreach)
> library(snow)
> library(doSNOW)
> cl <- makeCluster(2, type="SOCK")
> registerDoSNOW(cl)

> results <- hmm2(snp_exp)

> stopCluster(cl)
```

Citing this software

Robert B Scharpf, Giovanni Parmigiani, Jonathan Pevsner, and Ingo Ruczinski. Hidden Markov models for the assessment of chromosomal alterations using high-throughput SNP arrays. *Annals of Applied Statistics*, 2(2):687–713, 2008.

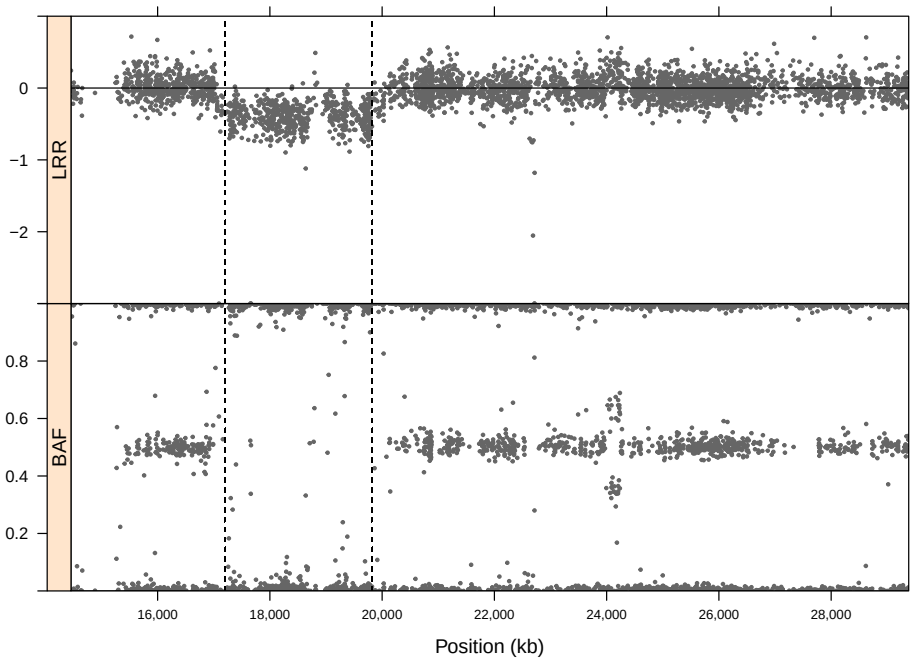


Figure 2: A hemizygous deletion (state 2) on chromosome 22 after merging adjacent hemizygous CNV calls.

Session Information

The version number of R and packages loaded for generating the vignette were:

- R version 4.0.0 (2020-04-24), `x86_64-w64-mingw32`
- Locale: `LC_COLLATE=C`, `LC_CTYPE=English_United States.1252`, `LC_MONETARY=English_United States.1252`, `LC_NUMERIC=C`, `LC_TIME=English_United States.1252`
- Running under: `Windows Server 2012 R2 x64 (build 9600)`
- Matrix products: default
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, stats4, utils
- Other packages: BSgenome 1.56.0, BSgenome.Hsapiens.UCSC.hg18 1.3.1000, Biobase 2.48.0, BiocGenerics 0.34.0, Biostrings 2.56.0, DelayedArray 0.14.0, GenomeInfoDb 1.24.0, GenomicRanges 1.40.0, IRanges 2.22.0, S4Vectors 0.26.0, SummarizedExperiment 1.18.0, VanillaICE 1.50.0, XVector 0.28.0, data.table 1.12.8, foreach 1.5.0, matrixStats 0.56.0, rtracklayer 1.48.0
- Loaded via a namespace (and not attached): BiocManager 1.30.10, BiocParallel 1.22.0, DBI 1.1.0, GenomeInfoDbData 1.2.3, GenomicAlignments 1.24.0, Matrix 1.2-18, RCurl 1.98-1.2, Rcpp 1.0.4.6, RcppEigen 0.3.3.7.0, Rsamtools 2.4.0, VGAM 1.1-2, XML 3.99-0.3, affyio 1.58.0, askpass 1.1, base64 2.0, beanplot 1.2, bit 1.1-15.2, bitops 1.0-6, codetools 0.2-16, compiler 4.0.0,

crayon 1.3.4, crlmm 1.46.0, ellipse 0.4.1, ff 2.2-14.2, grid 4.0.0, illuminaio 0.30.0, iterators 1.0.12, lattice 0.20-41, limma 3.44.0, mvtnorm 1.1-0, oligoClasses 1.50.0, openssl 1.4.1, preprocessCore 1.50.0, splines 4.0.0, tools 4.0.0, zlibbioc 1.34.0

References

[1] Terri H Beaty, Jeffrey C Murray, Mary L Marazita, Ronald G Munger, Ingo Ruczinski, Jacqueline B Hetmanski, Kung Yee Liang, Tao Wu, Tanda Murray, M. Daniele Fallin, Richard A Redett, Gerald Raymond, Holger Schwender, Sheng-Chih Jin, Margaret E Cooper, Martine Dunnwald, Maria A Mansilla, Elizabeth Leslie, Stephen Bullard, Andrew C Lidral, Lina M Moreno, Renato Menezes, Alexandre R Vieira, Aline Petrin, Allen J Wilcox, Rolv T Lie, Ethylin W Jabs, Yah Huei Wu-Chou, Philip K Chen, Hong Wang, Xiaoqian Ye, Shangzhi Huang, Vincent Yeow, Samuel S Chong, Sun Ha Jee, Bing Shi, Kaare Christensen, Mads Melbye, Kimberly F Doheny, Elizabeth W Pugh, Hua Ling, Eduardo E Castilla, Andrew E Czeizel, Lian Ma, L. Leigh Field, Lawrence Brody, Faith Pangilinan, James L Mills, Anne M Molloy, Peadar N Kirke, John M Scott, James M Scott, Mauricio Arcos-Burgos, and Alan F Scott. A genome-wide association study of cleft lip with and without cleft palate identifies risk variants near MAFB and ABCA4. *Nat Genet*, 42(6):525–529, Jun 2010.

[2] Robert B. Scharpf, Terri H. Beaty, Holger Schwender, Samuel G. Younkin, Alan F. Scott, and Ingo Ruczinski. Fast detection of de novo copy number variants from SNP arrays for case-parent trios. *BMC Bioinformatics*, 13(1):330, Dec 2012.

[3] Robert B Scharpf, Giovanni Parmigiani, Jonathan Pevsner, and Ingo Ruczinski. Hidden Markov models for the assessment of chromosomal alterations using high-throughput SNP arrays. *Annals of Applied Statistics*, 2(2):687–713, 2008.