The chromstaR user's guide

Aaron Taudt*

April 8, 2016

Contents

1 Introduction

ChIP-seq has become the standard technique for assessing the genome-wide chromatin state of DNA. *chromstaR* provides functions for the joint analysis of multiple ChIP-seq samples. It allows peak calling for transcription

*aaron.taudt@gmail.com

factor binding and histone modifications with a narrow (e.g. H3K4me3, H3K27ac, ...) or broad (e.g. H3K36me3, H3K27me3, ...) profile. All analysis can be performed on each sample individually (=univariate), or in a joint analysis considering all samples simultaneously (=multivariate).

2 Outline of workflow

Every analysis with the *chromstaR* package starts from aligned reads in either BAM or BED format. In the first step, the genome is partitioned into non-overlapping, equally sized bins and the reads that fall into each bin are counted. These read counts serve as the basis for both the univariate and the multivariate peak- and broad-region calling. Univariate peak calling is done by fitting a three-state Hidden Markov Model to the binned read counts. Multivariate peak calling for $\mathcal{S}$ samples is done by fitting a $2^{\mathcal{S}}$-state Hidden Markov Model to all binned read counts.

3 Univariate analysis

3.1 Task 1: Peak calling for a narrow histone modification

Examples of histone modifications with a narrow profile are H3K4me3, H3K9ac and H3K27ac in most human tissues. For such peak-like modifications, the bin size should be set to a value between 200bp and 1000bp.

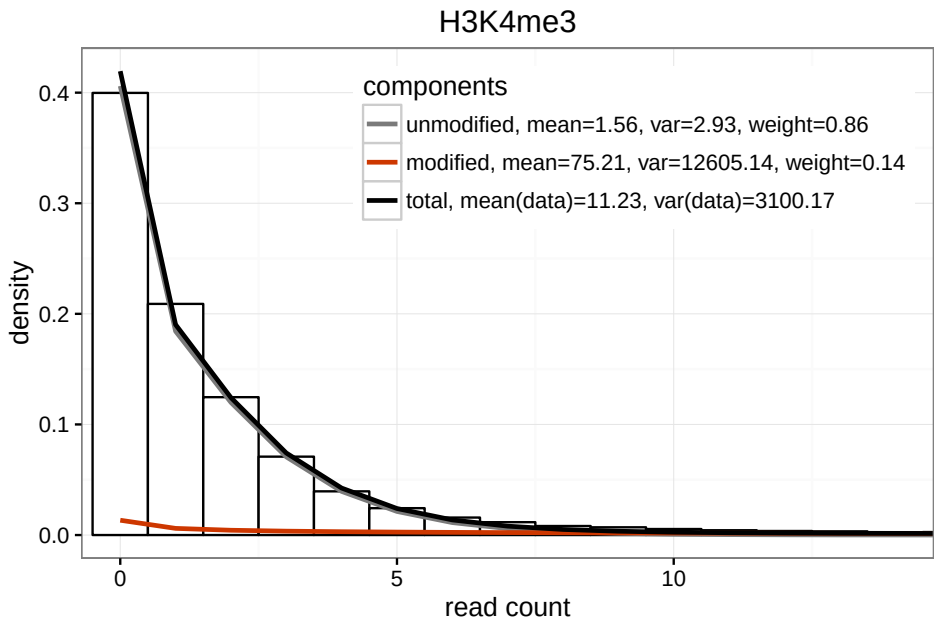
```
library(chromstaR)
```

```
## === Step 1: Binning ===
# Get an example BED file
bedfile <- system.file("extdata","euratrans","liver-H3K4me3-BN-male-bio2-tech1.bed.gz",
                       package="chromstaRData")
# Get the chromosome lengths (see ?GenomeInfoDb::fetchExtendedChromInfoFromUCSC)
# This is only necessary for BED files. BAM files are handled automatically.
chrom.lengths <- data(rn4_chrominfo)
# We use bin size 1000bp and chromosome 12 to keep the example quick
binned.data <- binReads(bedfile, assembly=rn4_chrominfo, binsizes=1000,
                        chromosomes='chr12')
print(binned.data)

## GRanges object with 46782 ranges and 1 metadata column:
##           seqnames               ranges strand |    counts
##              <Rle>            <IRanges>  <Rle> | <integer>
##       [1]    chr12         [   1, 1000]      * |         0
##       [2]    chr12         [1001, 2000]      * |         0
##       [3]    chr12         [2001, 3000]      * |         0
##       [4]    chr12         [3001, 4000]      * |         0
```

```
##       [5]    chr12         [4001, 5000]      * |         1
##       ...      ...                  ...    ... .       ...
##   [46778]    chr12 [46777001, 46778000]      * |         1
##   [46779]    chr12 [46778001, 46779000]      * |         4
##   [46780]    chr12 [46779001, 46780000]      * |         2
##   [46781]    chr12 [46780001, 46781000]      * |         2
##   [46782]    chr12 [46781001, 46782000]      * |         1
##   -------
##   seqinfo: 1 sequence from an unspecified genome
```

```
## === Step 2: Peak calling ===
# We restrict the peak calling to 60 seconds to keep this example quick.
model <- callPeaksUnivariate(binned.data, ID='H3K4me3', max.time=60, verbosity=0)

## Replaced read counts > 500 by 500 in 331 bins.  Set option 'read.cutoff=FALSE' to disable
this filtering.  This filtering was done to increase the speed of the HMM.
## Calculating states from posteriors ...
##  0.12s
## Making segmentation ...
##  0.3s
```

```
## === Step 3: Checking the fit ===
# For a narrow modification, the fit should look something like this,
# with the 'modified'-component near the bottom of the figure
plot(model)
```



```
## === Step 4: Export to genome browser ===
# We can export peak calls and binned read counts with
exportUnivariates(list(model), filename='your-peak-file', what='peaks')
exportUnivariates(list(model), filename='your-read-file', what='counts')
```

3.2 Task 2: Peak calling for a broad histone modification

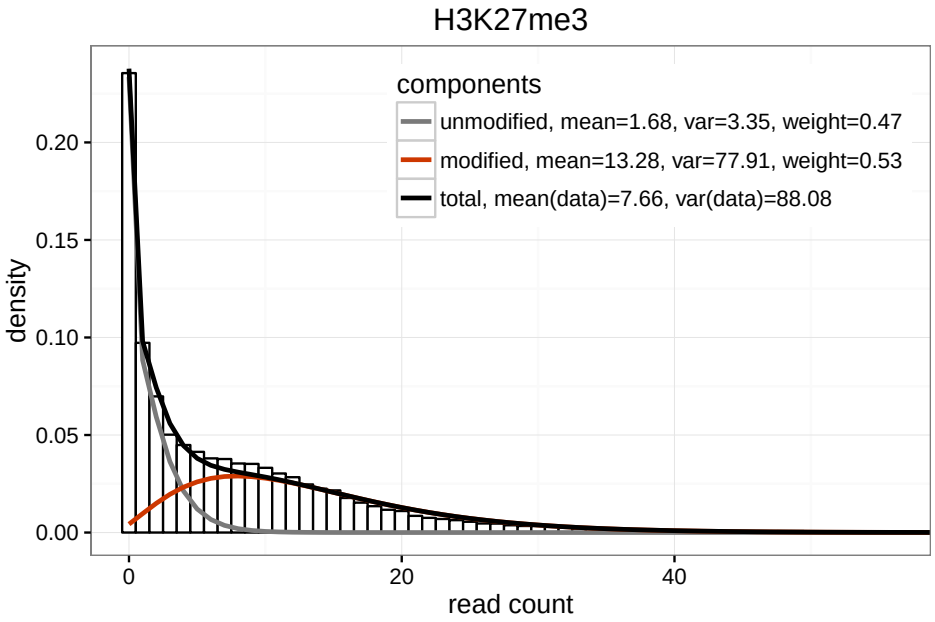
Examples of histone modifications with a broad profile are H3K9me3, H3K27me3, H3K36me3, H4K20me1 in most human tissues. These modifications usually cover broad domains of the genome, and the enrichment is best captured with a bin size between 500bp and 2000bp.

```
library(chromstaR)
```

```
## === Step 1: Binning ===
# Get an example BED file
bedfile <- system.file("extdata","euratrans","liver-H3K27me3-BN-male-bio1-tech1.bed.gz",
                       package="chromstaRData")
# Get the chromosome lengths (see ?GenomeInfoDb::fetchExtendedChromInfoFromUCSC)
# This is only necessary for BED files. BAM files are handled automatically.
chrom.lengths <- data(rn4_chrominfo)
# We use bin size 1000bp and chromosome 12 to keep the example quick
binned.data <- binReads(bedfile, assembly=rn4_chrominfo, binsizes=1000,
                       chromosomes='chr12')
print(binned.data)
```

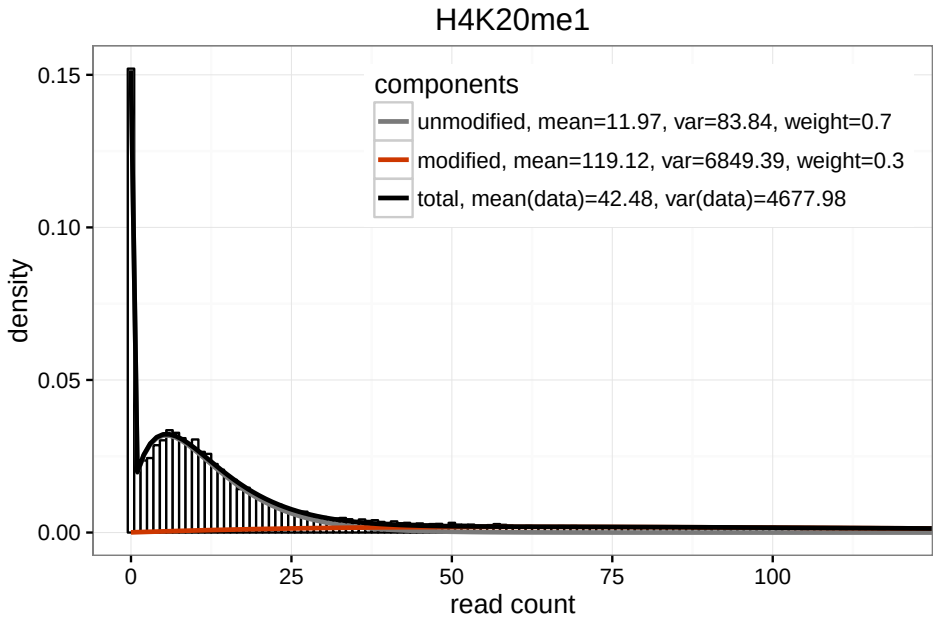
```
## === Step 2: Peak calling ===
# We restrict the peak calling to 60 seconds to keep this example quick.
model <- callPeaksUnivariate(binned.data, ID='H3K27me3', max.time=60, verbosity=0)

## Calculating states from posteriors ...
##  0.1s
## Making segmentation ...
##  0.15s
```

```
## === Step 3: Checking the fit ===
# For a broad modification, the fit should look something like this,
# with a 'modified'-component that fits the thick tail of the distribution.
plot(model)
```




```
## === Step 4: Export to genome browser ===
# We can export peak calls and binned read counts with
exportUnivariates(list(model), filename='your-peak-file', what='peaks')
exportUnivariates(list(model), filename='your-read-file', what='counts')
```

```
## === Step 1-3: Another example for mark H4K20me1 ===
bedfile <- system.file("extdata","euratrans","liver-H4K20me1-BN-male-bio1-tech1.bed.gz",
                       package="chromstaRData")
data(rn4_chrominfo)
binned.data <- binReads(bedfile, assembly=rn4_chrominfo, binsizes=1000,
                        chromosomes='chr12')
model <- callPeaksUnivariate(binned.data, ID='H4K20me1', max.time=60, verbosity=0)
plot(model)
```



3.3 Task 3: Peak calling for ATAC-seq, DNase-seq, FAIRE-seq, ...

Peak calling for ATAC-seq and DNase-seq is similar to the peak calling of a narrow histone modification (see section 3.1). FAIRE-seq experiments seem to exhibit a broad profile with our model, so the procedure is similar to the domain calling of a broad histone modification (see section 3.2).

4 Multivariate analysis

4.1 Task 1: Integrating multiple replicates

chromstaR can be used to call peaks with multiple replicates, without the need of prior merging. The underlying statistical model integrates information from all replicates to identify common peaks. It is, however, important to note that replicates with poor quality can affect the joint peak calling negatively. It is therefore recommended to first check the replicate quality and discard poor-quality replicates. The necessary steps for peak calling for an example ChIP-seq experiment with 4 replicates are detailed below.

```
library(chromstaR)
```

```r
## === Step 1: Binning ===
# Let's get some example data with 3 replicates
file.path <- system.file("extdata","euratrans", package='chromstaRData')
bedfiles.good <- list.files(file.path, pattern="liver.*H3K27me3", full.names=TRUE)[1:3]
# We fake a replicate with poor quality by taking a different mark entirely
bedfiles.poor <- list.files(file.path, pattern="liver.*H4K20me1", full.names=TRUE)[1]
bedfiles <- c(bedfiles.good, bedfiles.poor)
# This is only necessary for BED files. BAM files are handled automatically.
data(rn4_chrominfo)
# We use bin size 1000bp and chromosome 12 to keep the example quick
binned.data <- list()
for (bedfile in bedfiles) {
  binned.data[[basename(bedfile)]] <- binReads(bedfile, binsize=1000,
                                       assembly=rn4_chrominfo, chromosomes='chr12')
}
```

```r
## === Step 2: Univariate peak calling ===
# The univariate fit is obtained for each replicate
models <- list()
for (i1 in 1:length(binned.data)) {
  models[[i1]] <- callPeaksUnivariate(binned.data[[i1]], ID=paste0('Rep',i1),
                                      max.time=60)
}
```

```r
## === Step 3: Check replicate correlation ===
# We run a multivariate peak calling on all 4 replicates
# A warning is issued because replicate 4 is very different from the others
multi.model <- callPeaksReplicates(models, max.time=60, eps=1)

## HMM: number of states = 16
## HMM: number of bins = 46782
## HMM: maximum number of iterations = none
## HMM: maximum running time = 60 sec
## HMM: epsilon = 1
## HMM: number of experiments = 4
##  Iteration              log(P)             dlog(P)    Time in sec
##          0                -inf                   -              0
## HMM: Precomputing densities ...
##  Iteration              log(P)             dlog(P)    Time in sec
##          0                -inf                   -              0
##          1      -555080.685888                inf              0
##          2      -550061.722635         5018.963253              1
##          3      -549932.374163          129.348472              1
##          4      -549910.169776           22.204388              1
##          5      -549903.227764            6.942011              1
##          6      -549900.149552            3.078213              1
##          7      -549898.485193            1.664359              1
##          8      -549897.382329            1.102864              1
##          9      -549896.538227            0.844102              1
## HMM: Convergence reached!
## HMM: Recoding posteriors ...

## Warning in callPeaksReplicates(models, max.time = 60, eps = 1):  Your replicates cluster
in 2 groups.  Consider redoing your analysis with only the group with the highest average
```

```
coverage:
## Rep4
## Replicates from groups with lower coverage are:
## Rep1
## Rep2
## Rep3

# Checking the correlation confirms that Rep4 is very different from the others
print(multi.model$replicateInfo$correlation)
```

```
## === Step 4: Peak calling with replicates ===
# We redo the previous step without the "bad" replicate
# Also, we force all replicates to agree in their peak calls
multi.model <- callPeaksReplicates(models[1:3], force.equal=TRUE, max.time=60)
```

```
## === Step 5: Export to genome browser ===
# Finally, we can export the results as BED file
exportMultivariate(multi.model, filename='your-peak-file', what='peaks')
exportMultivariate(multi.model, filename='your-read-file', what='counts')
```

4.2 Task 2: Detecting differentially modified regions

chromstaR is extremely powerful in detecting differentially modified regions in two or more samples. The following example illustrates this on ChIP-seq data for H4K20me1 in liver and heart (left-ventricle) tissues from rat. The mode of analysis is called "condition", because all conditions are analyzed simultaneously.

```
library(chromstaR)
```

```
#=== Step 1: Preparation ===
## Prepare the file paths. Exchange this with your input and output directories.
inputfolder <- system.file("extdata","euratrans", package="chromstaRData")
outputfolder <- file.path(tempdir(), 'H4K20me1-example')

## Define experiment structure
data(experiment_table_H4K20me1)
print(experiment_table_H4K20me1)
```

```
##                                      file     mark condition replicate
## 1 liver-H4K20me1-BN-male-bio1-tech1.bed.gz H4K20me1     liver         1
## 2 liver-H4K20me1-BN-male-bio2-tech1.bed.gz H4K20me1     liver         2
## 3    lv-H4K20me1-BN-male-bio1-tech1.bed.gz H4K20me1        lv         1
## 4    lv-H4K20me1-BN-male-bio2-tech1.bed.gz H4K20me1        lv         2
##   pairedEndReads
## 1          FALSE
## 2          FALSE
## 3          FALSE
## 4          FALSE
```

```
## Define assembly
# This is only necessary if you have BED files, BAM files are handled automatically.
# For common assemblies you can also specify them as 'hg19' for example.
data(rn4_chrominfo)
head(rn4_chrominfo)
```

```
##   UCSC_seqlevel UCSC_seqlength NCBI_seqlevel
## 1          chrM          16300            MT
## 2         chr12       46782294            12
## 3         chr20       55268282            20
## 4         chr19       59218465            19
## 5         chr18       87265094            18
## 6         chr11       87759784            11
```

```
#=== Step 2: Run Chromstar ===
## Run ChromstaR
Chromstar(inputfolder, experiment.table=experiment_table_H4K20me1,
          outputfolder=outputfolder, numCPU=2, binsize=1000, assembly=rn4_chrominfo,
          prefit.on.chr='chr12', mode='condition')
```

```
## Results are stored in 'outputfolder' and can be loaded for further processing
list.files(outputfolder)
```

```
## [1] "binned"           "browserfiles"     "chrominfo.tsv"
## [4] "chromstaR.config" "multivariate"     "univariate"
```

```
model <- get(load(file.path(outputfolder,'multivariate',
                            'multivariate_mode-condition_mark-H4K20me1.RData')))
```

```
## === Step 3: Construct differential and common states ===
diff.states <- stateBrewer(experiment_table_H4K20me1, mode='condition',
                           differential.states=TRUE)
print(diff.states)
```

```
##   combination state
## 1        [lv]     3
## 2     [liver]    12
```

```
common.states <- stateBrewer(experiment_table_H4K20me1, mode='condition',
                             common.states=TRUE)
print(common.states)
```

```
##   combination state
## 1          []     0
## 2  [liver+lv]    15
```

```
## === Step 5: Export to genome browser ===
# Export only differential states
exportMultivariate(multi.model, filename='your-peak-file', what='peaks',
                   include.states=diff.states)
exportMultivariate(multi.model, filename='your-read-file', what='counts',
                   include.states=diff.states)
exportMultivariate(multi.model, filename='your-combstates-file', what='combinations',
                   include.states=diff.states)
```

4.3 Task 3: Finding combinatorial chromatin states

Most experimental studies that probe several histone modifications are interested in combinatorial chromatin states. An example of a simple combinatorial state would be [H3K4me3+H3K27me3], which is also frequently called "bivalent promoter", due to the simultaneous occurrence of the promoter marking H3K4me3 and the repressive H3K27me3. Finding combinatorial states with *chromstaR* is equivalent to a multivariate peak calling. The following code chunks demonstrate how to find bivalent promoters and do some simple analysis:

```
library(chromstaR)
```

```
#=== Step 1: Preparation ===
## Prepare the file paths. Exchange this with your input and output directories.
inputfolder <- system.file("extdata","euratrans", package="chromstaRData")
outputfolder <- file.path(tempdir(), 'liver-example')

## Define experiment structure
data(experiment_table_liver)
print(experiment_table_liver)
```

```
##                                     file     mark condition replicate
## 1 liver-H3K27me3-BN-male-bio1-tech1.bed.gz H3K27me3     liver         1
## 2 liver-H3K27me3-BN-male-bio1-tech2.bed.gz H3K27me3     liver         2
## 3  liver-H3K4me3-BN-male-bio2-tech1.bed.gz  H3K4me3     liver         2
## 4  liver-H3K4me3-BN-male-bio3-tech1.bed.gz  H3K4me3     liver         3
##   pairedEndReads
## 1          FALSE
## 2          FALSE
## 3          FALSE
## 4          FALSE
```

```
## Define assembly
# This is only necessary if you have BED files, BAM files are handled automatically.
# For common assemblies you can also specify them as 'hg19' for example.
data(rn4_chrominfo)
head(rn4_chrominfo)

##   UCSC_seqlevel UCSC_seqlength NCBI_seqlevel
## 1          chrM          16300            MT
## 2         chr12       46782294            12
## 3         chr20       55268282            20
## 4         chr19       59218465            19
## 5         chr18       87265094            18
## 6         chr11       87759784            11
```

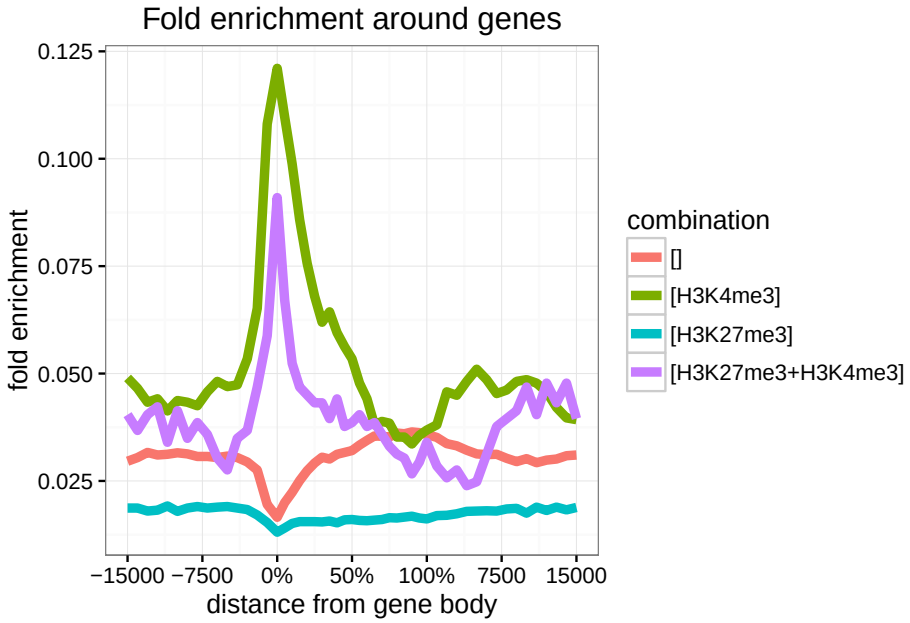
```
#=== Step 2: Run Chromstar ===
## Run ChromstaR
Chromstar(inputfolder, experiment.table=experiment_table_liver,
          outputfolder=outputfolder, numCPU=2, binsize=1000, assembly=rn4_chrominfo,
          prefit.on.chr='chr12', mode='mark')
```

```
## Results are stored in 'outputfolder' and can be loaded for further processing
list.files(outputfolder)

## [1] "binned"           "browserfiles"     "chrominfo.tsv"
## [4] "chromstaR.config" "multivariate"     "univariate"

model <- get(load(file.path(outputfolder,'multivariate',
                            'multivariate_mode-mark_condition-liver.RData')))
```

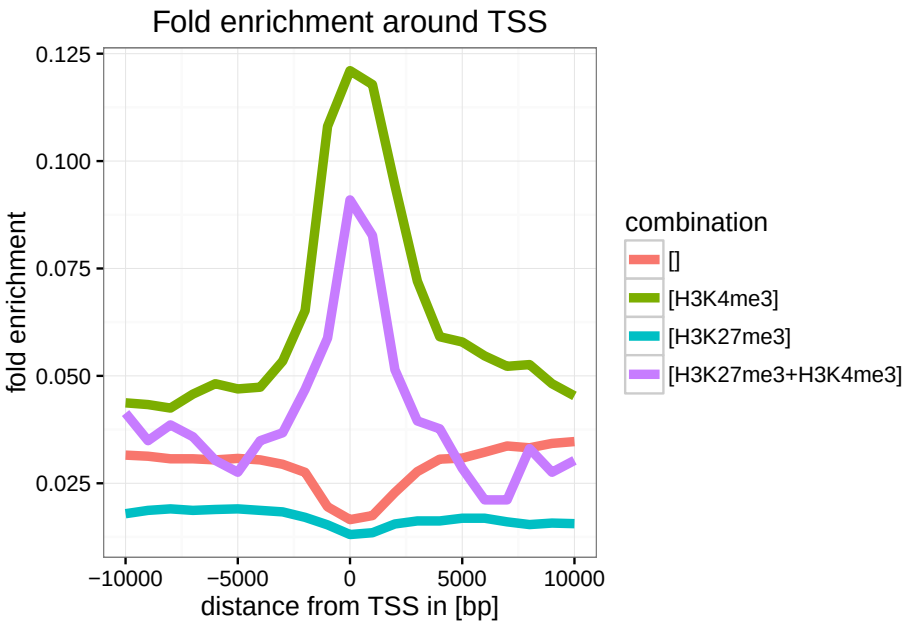
```
## === Step 3: Enrichment analysis ===
library(biomaRt)
ensembl <- useMart('ENSEMBL_MART_ENSEMBL', host='may2012.archive.ensembl.org',
                   dataset='rnorvegicus_gene_ensembl')
genes <- getBM(attributes=c('ensembl_gene_id', 'chromosome_name', 'start_position',
                            'end_position', 'strand', 'external_gene_id'),
               mart=ensembl)
# Transform to GRanges for easier handling
genes <- GRanges(seqnames=paste0('chr',genes$chrom),
                 ranges=IRanges(start=genes$start, end=genes$end),
                 strand=genes$strand,
                 name=genes$external_gene_id)
print(genes)

## GRanges object with 29516 ranges and 1 metadata column:
##           seqnames                 ranges strand |        name
##              <Rle>              <IRanges>  <Rle> | <character>
##       [1]    chr13     [1120899, 1121213]      - |   LOC682397
##       [2]    chr13     [1192186, 2293551]      - |   LOC304725
##       [3]    chr13     [3174383, 3175216]      + |
##       [4]    chr13     [4377731, 4379174]      - |  D3ZPH4_RAT
##       [5]    chr13     [4866302, 4866586]      - |  F1LZC7_RAT
```

```
##       ...      ...                       ...    ... .         ...
##   [29512]     chr6 [134310258, 134310338]      + |    SNORD113
##   [29513]     chr9 [  6920889,   6921049]      - |          U1
##   [29514]    chr11 [ 40073746,  40073816]      - |    SNORD19B
##   [29515]     chr2 [233090372, 233090478]      - |          U6
##   [29516]     chr6 [ 92917449,  92917541]      + |
##   -------
##   seqinfo: 22 sequences from an unspecified genome; no seqlengths
```

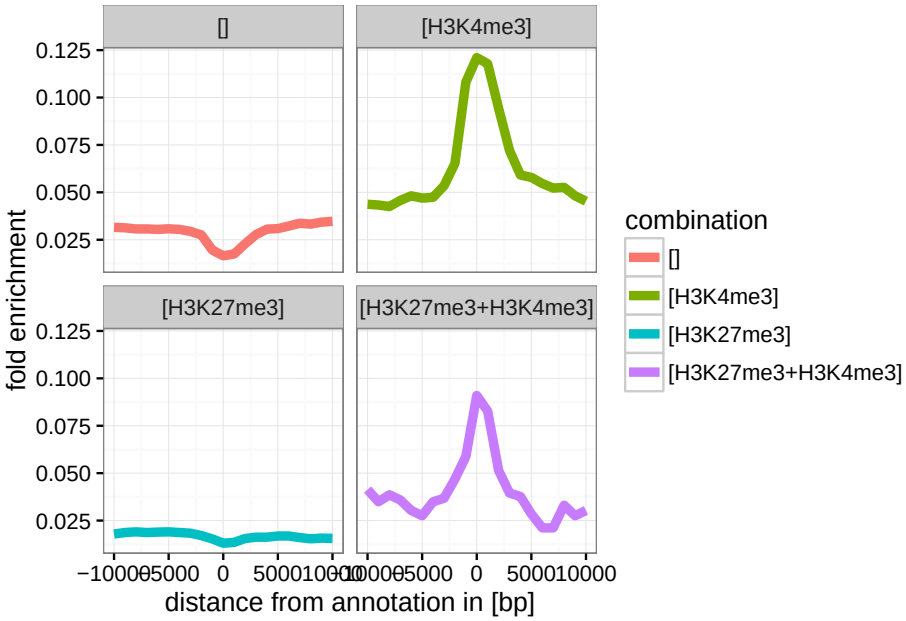
```
# We expect promoter [H3K4me3] and bivalent-promoter signatures [H3K4me3+H3K27me3]
# to be enriched at transcription start sites.
plotEnrichment(hmm = model, annotation = genes, bp.around.annotation = 15000) +
  ggtitle('Fold enrichment around genes') +
  xlab('distance from gene body')
```



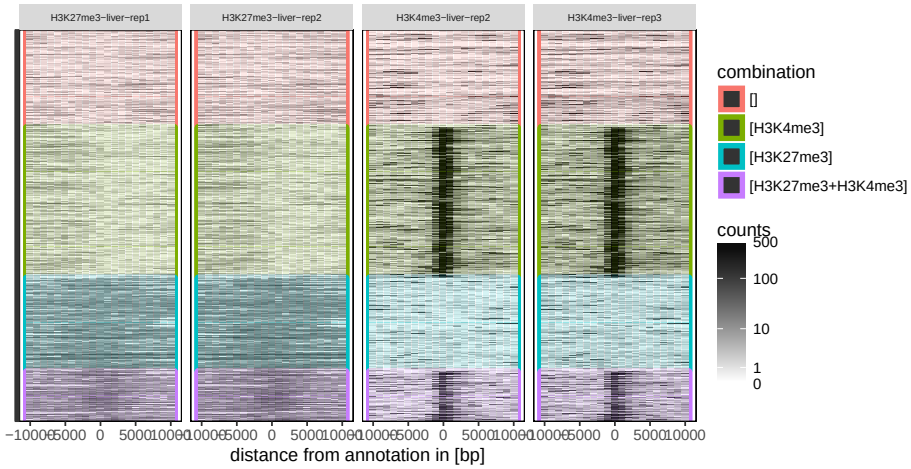

```
# Plot enrichment only at TSS. We make use of the fact that TSS is the start of a gene.
plotEnrichment(model, genes, region = 'start') +
  ggtitle('Fold enrichment around TSS') +
  xlab('distance from TSS in [bp]')
```




```
# Note: If you want to facet the plot because you have many combinatorial states you
# can do that with
plotEnrichment(model, genes, region = 'start') +
  facet_wrap(~ combination)
```



```
# Another form of visualization that shows every TSS in a heatmap
tss <- resize(genes, width = 3, fix = 'start')
plotHeatmap(model, tss) +
  theme(strip.text.x = element_text(size=6))
```



4.4 Task 4: Finding differences between combinatorial chromatin states

Consider bivalent promoters defined by [H3K4me3+H3K27me3] at two different developmental stages. This is an example where one is interested in *differences* between *combinatorial states*. The following example demonstrates how such an analysis can be done with *chromstaR*. We use the same data set as before with bivalent promoters in heart (left-ventricle) and liver tissue.

`Chromstar` can be run in three different modes:

- *full*: Recommended mode if your (number of marks) * (number of conditions) is less or equal to 8. With 8 ChIP-seq experiments there are already $2^8 = 256$ combinatorial states which is the maximum that most computers can handle computationally for a human-sized genome at bin size 1000bp.

- *condition*: Choose this mode if you are interested in highly significant differences between conditions. The computational limit for the number of conditions is $\sim$ 8 for a human-sized genome. Combinatorial states are not as accurate as in mode *mark* or *full*.

- **DEFAULT** *mark*: This mode will yield good combinatorial chromatin state calls for any number of marks and conditions. However, differences between conditions have more false positives than in mode *condition* or *full*.

```
library(chromstaR)
```

```
#=== Step 1: Preparation ===
## Prepare the file paths. Exchange this with your input and output directories.
inputfolder <- system.file("extdata","euratrans", package="chromstaRData")
outputfolder <- file.path(tempdir(), 'liver-lv-example')

## Define experiment structure
data(experiment_table)
print(experiment_table)
```

```
##                                       file     mark condition replicate
## 1  liver-H3K27me3-BN-male-bio1-tech1.bed.gz H3K27me3     liver         1
## 2  liver-H3K27me3-BN-male-bio1-tech2.bed.gz H3K27me3     liver         2
## 3   liver-H3K4me1-BN-male-bio1-tech1.bed.gz  H3K4me1     liver         1
## 4   liver-H3K4me1-BN-male-bio2-tech1.bed.gz  H3K4me1     liver         2
## 5   liver-H3K4me3-BN-male-bio2-tech1.bed.gz  H3K4me3     liver         2
## 6   liver-H3K4me3-BN-male-bio3-tech1.bed.gz  H3K4me3     liver         3
## 7  liver-H4K20me1-BN-male-bio1-tech1.bed.gz H4K20me1     liver         1
## 8  liver-H4K20me1-BN-male-bio2-tech1.bed.gz H4K20me1     liver         2
## 9     lv-H3K27me3-BN-male-bio2-tech1.bed.gz H3K27me3        lv         1
## 10    lv-H3K27me3-BN-male-bio2-tech2.bed.gz H3K27me3        lv         2
## 11     lv-H3K4me1-BN-male-bio1-tech1.bed.gz  H3K4me1        lv         1
## 12     lv-H3K4me1-BN-male-bio2-tech1.bed.gz  H3K4me1        lv         2
## 13   lv-H3K4me3-BN-female-bio1-tech1.bed.gz  H3K4me3        lv         1
## 14     lv-H3K4me3-BN-male-bio2-tech1.bed.gz  H3K4me3        lv         2
## 15    lv-H4K20me1-BN-male-bio1-tech1.bed.gz H4K20me1        lv         1
## 16    lv-H4K20me1-BN-male-bio2-tech1.bed.gz H4K20me1        lv         2
##    pairedEndReads
## 1           FALSE
## 2           FALSE
## 3           FALSE
## 4           FALSE
## 5           FALSE
## 6           FALSE
## 7           FALSE
## 8           FALSE
## 9           FALSE
## 10          FALSE
## 11          FALSE
## 12          FALSE
## 13          FALSE
## 14          FALSE
## 15          FALSE
## 16          FALSE
```

```
## Define assembly
```

```
# This is only necessary if you have BED files, BAM files are handled automatically.
# For common assemblies you can also specify them as 'hg19' for example.
data(rn4_chrominfo)
head(rn4_chrominfo)
```

```
##   UCSC_seqlevel UCSC_seqlength NCBI_seqlevel
## 1          chrM          16300            MT
## 2         chr12       46782294            12
## 3         chr20       55268282            20
## 4         chr19       59218465            19
## 5         chr18       87265094            18
## 6         chr11       87759784            11
```

```
#=== Step 2: Run Chromstar ===
## Run ChromstaR
Chromstar(inputfolder, experiment.table=experiment_table,
          outputfolder=outputfolder, numCPU=2, binsize=1000, assembly=rn4_chrominfo,
          prefit.on.chr='chr12', mode='mark')
```

```
## Results are stored in 'outputfolder' and can be loaded for further processing
list.files(outputfolder)
```

```
## [1] "binned"           "browserfiles"     "chrominfo.tsv"
## [4] "chromstaR.config" "combined"         "multivariate"
## [7] "univariate"
```

```
model <- get(load(file.path(outputfolder,'multivariate',
                            'multivariate_mode-mark_condition-liver.RData')))
```

5 FAQ

6 Session Info

```
sessionInfo()
```

```
## R Under development (unstable) (2016-02-17 r70182)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 14.04.4 LTS
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=nl_NL.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=nl_NL.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=nl_NL.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=nl_NL.UTF-8 LC_IDENTIFICATION=C
##
```

```
## attached base packages:
## [1] stats4    parallel  stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] biomaRt_2.27.2        chromstaR_0.98.0      chromstaRData_0.99.0
##  [4] ggplot2_2.1.0         GenomicRanges_1.23.25 GenomeInfoDb_1.7.6
##  [7] IRanges_2.5.42        S4Vectors_0.9.46      BiocGenerics_0.17.4
## [10] knitr_1.12.3          devtools_1.10.0
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.4                 compiler_3.3.0
##  [3] formatR_1.3                 plyr_1.8.3
##  [5] highr_0.5.1                 XVector_0.11.8
##  [7] bitops_1.0-6                iterators_1.0.8
##  [9] tools_3.3.0                 zlibbioc_1.17.1
## [11] digest_0.6.9                RSQLite_1.0.0
## [13] evaluate_0.8.3              memoise_1.0.0
## [15] gtable_0.2.0                foreach_1.4.3
## [17] DBI_0.3.1                   stringr_1.0.0
## [19] Biostrings_2.39.12          grid_3.3.0
## [21] Biobase_2.31.3              AnnotationDbi_1.33.7
## [23] XML_3.98-1.4                BiocParallel_1.5.21
## [25] reshape2_1.4.1              magrittr_1.5
## [27] scales_0.4.0                Rsamtools_1.23.7
## [29] codetools_0.2-14            GenomicAlignments_1.7.20
## [31] SummarizedExperiment_1.1.23 colorspace_1.2-6
## [33] labeling_0.3                stringi_1.0-1
## [35] RCurl_1.95-4.8              munsell_0.4.3
## [37] doParallel_1.0.10

warnings()

## NULL
```