Integrative Analysis of Epigenomic sequencing (and microarray) data with `Repitools`

Mark Robinson Aaron Statham Dario Strbenac

Last compiled on: April 30, 2018

1 Introduction

`Repitools` is a package that allows exploratory as well as targeted statistical analysis of absolute and differential binding for ChIP-seq and MeDIP-seq data types, and gives visual summaries in a variety of formats. Some basic quality checking utilities are available for sequencing data. Much of the functionality available is implemented for both tiling microarrays and sequencing data, with very similar function calls for both types of data.

In this vignette, we highlight various features within the package. Further description of the package can be found in the associated Bioinformatics Applications Note[1] as well as in the help documents.

To start with, set a random seed, and load the `Repitools` package:

```
> options(prompt = " ", continue = " ")
 set.seed(4)
 library(Repitools)
```

2 Example Datasets

Caution : All data distributed with the package is a reduced version of the full dataset presented here. Also, all variable names have `.subset` appended to them. Therefore, the code presented here cannot simply be copied and pasted into **R**.

A small `GRangesList` of mapped short reads (four samples run on an Illumina Genome Analyser) is included with the package (for example, see `?binPlots`). This data has been published and is available here. LNCaP is a prostate cancer cell line, and PrEC is a (normal) prostate epithelial cell line. Here, the "IP" represents an MBD capture experiment, whereby a population of DNA fragments containing methylated DNA (generally in the CpG context) and "input" represents fragmented genomic DNA from the same cell lines.

Note that `GRanges` objects of mapped reads from many popular aligners can be created in **R** using the `readAligned` function in the `ShortRead` package, then coerced with `as(alnRdObj,`

[1] Repitools: an R package for the analysis of enrichment-based epigenomic data

"GRanges"). Alternatively, two convenience methods `BAM2GRanges` and `BAM2GRangesList` in `Repitools` could also be used, if the reads were stored on disk in BAM format (this uses the `scanBam` function from the `Rsamtools` package). By default, these two methods read in only the uniquely-mapping reads. See the `ShortRead` package documentation for ideas about how to read other sequencing data into `GRanges` or `GRangesList` objects.

```
 library(GenomicRanges)
 data(samplesList)
 class(samples.list)
```

```
[1] "GRangesList"
attr(,"package")
[1] "GenomicRanges"
```

```
 names(samples.list)
```

```
[1] "PrEC input"  "PrEC IP"     "LNCaP input" "LNCaP IP"
```

```
 elementNROWS(samples.list)
```

```
 PrEC input     PrEC IP LNCaP input    LNCaP IP
   11061279    10008129    19119904    10139044
```

```
 samples.list[[1]]
```

```
GRanges with 11061279 ranges and 0 elementMetadata values:
             seqnames          ranges strand
                <Rle>       <IRanges>  <Rle>
         [1]     chr1   [ 248,  283]      +
         [2]     chr1   [ 447,  482]      -
         [3]     chr1   [ 602,  637]      -
         [4]     chr1   [3182, 3217]      +
         [5]     chr1   [4783, 4818]      -
         [6]     chr1   [6287, 6322]      -
         [7]     chr1   [6310, 6345]      +
         [8]     chr1   [7340, 7375]      -
         [9]     chr1   [9103, 9138]      -
         ...      ...             ...    ...
  [11061271]     chrM [16531, 16566]      +
  [11061272]     chrM [16532, 16567]      +
  [11061273]     chrM [16532, 16567]      -
  [11061274]     chrM [16533, 16568]      -
  [11061275]     chrM [16533, 16568]      +
  [11061276]     chrM [16534, 16569]      -
  [11061277]     chrM [16535, 16570]      -
  [11061278]     chrM [16536, 16571]      +
  [11061279]     chrM [16536, 16571]      -
  ---
  seqlengths:
        chr1      chr2      chr3      chr4 ...       chrX      chrY      chrM
   247249719 242951149 199501827 191273063 ... 154913754  57772954     16571
```

Also, an annotation of genes will be used. The annotation used here is based on one provided from Affymetrix for their Gene 1.0 ST expression arrays[2]. We will relate the epigenomic sequencing data to the Affymetrix gene expression measurements. Of course, users may wish to make use of the rich functionality available within the `GenomicFeatures` package.

```
gene.anno <- read.csv(system.file("data/geneAnno.csv", package = "Repitools"),
                      stringsAsFactors = FALSE)
head(gene.anno)
```

```
     name  chr strand  start    end    symbol
1 7896759 chr1      + 781253 783614 LOC643837
2 7896761 chr1      + 850983 869824    SAMD11
3 7896779 chr1      + 885829 890958    KLHL17
4 7896798 chr1      + 891739 900345   PLEKHN1
5 7896817 chr1      + 938709 939782     ISG15
6 7896822 chr1      + 945365 981355      AGRN
```

```
dim(gene.anno)
```

```
[1] 24966     6
```

Lastly, there is matrix of gene expression changes, with each element related to the corresponding row in the gene annotation table. These values are moderated t-statistics (see the `limma` package) of background corrected and RMA normalised Affymetrix expression array experiments. The unprocessed array data is available here.

```
data(expr)
head(expr)
```

```
           t-stat
7896759  4.1130688
7896761  3.0691214
7896779  0.9724271
7896798 -0.5090460
7896817  2.1949896
7896822 -6.4049774
```

```
dim(expr)
```

```
[1] 24966     1
```

3 Quality Checking

As mentioned, two of the samples are MBD2 IPs, and two are inputs. Therefore, the IP samples should differ to the inputs in at least two ways. Firstly, they should be more CpG-rich, since we are enriching for methylated DNA, whcih rarely occurs outside of this sequence context. Secondly, DNA methylation tends to occur in peaks since CpG sites are often present in CpG-rich islands.

[2] http://www.affymetrix.com/Auth/analysis/downloads/na27/wtgene/HuGene-1_0-st-v1.na27.hg18.transcript.csv.zip

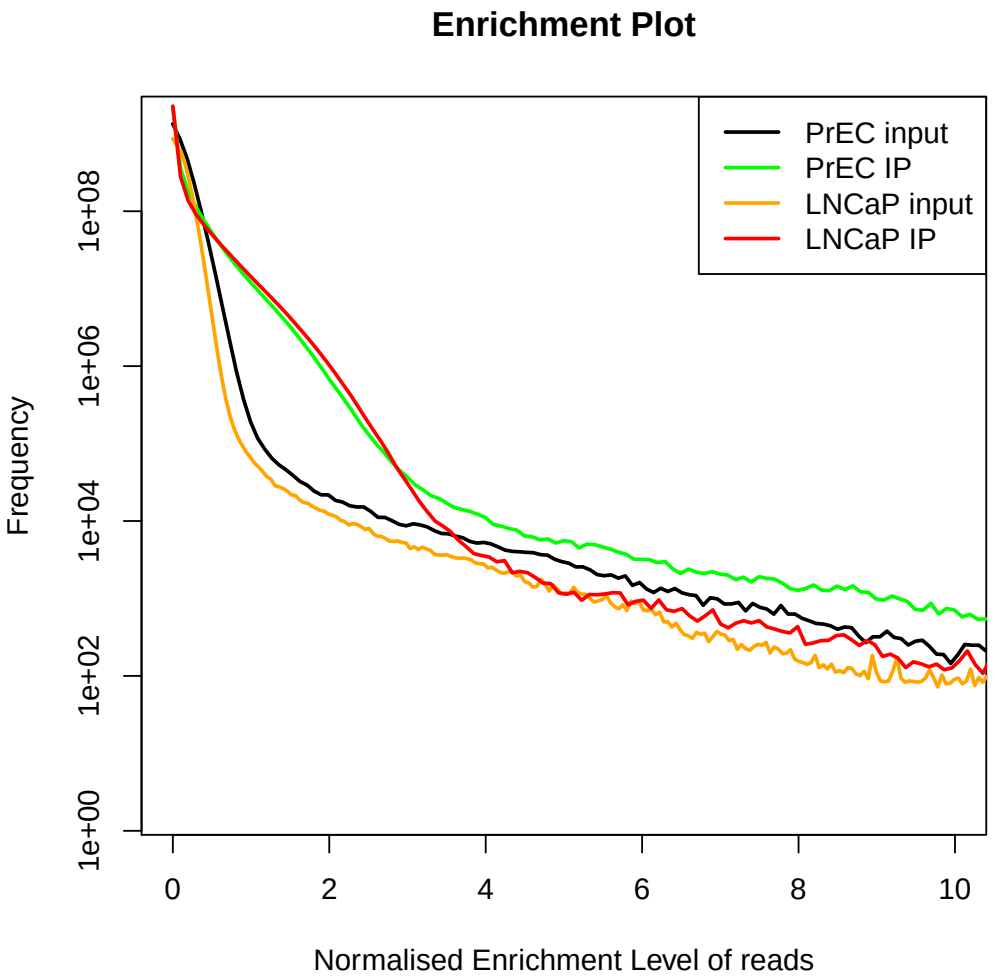
Conversely, the input samples should be distributed somewhat uniform genome-wide, aside from the usual mappability and GC content biases.

We can visualize the (log) frequencies of normalized coverage to get an idea of whether the reads occur in clusters or more dispersed, at least in a relative sense. For this, we can use `enrichmentPlot`. Similarly, we can calculate the CpG density of reads (or reads extended to a certain fragment size) and plot distributions across multiple samples using `cpgDensityPlot`, as below.

```
seqinfo(samples.list)
```

```
Seqinfo of length 25
seqnames seqlengths isCircular genome
chr1      247249719       <NA>   hg18
chr2      242951149       <NA>   hg18
chr3      199501827       <NA>   hg18
chr4      191273063       <NA>   hg18
chr5      180857866       <NA>   hg18
chr6      170899992       <NA>   hg18
chr7      158821424       <NA>   hg18
chr8      146274826       <NA>   hg18
chr9      140273252       <NA>   hg18
...             ...        ...    ...
chr17      78774742       <NA>   hg18
chr18      76117153       <NA>   hg18
chr19      63811651       <NA>   hg18
chr20      62435964       <NA>   hg18
chr21      46944323       <NA>   hg18
chr22      49691432       <NA>   hg18
chrX      154913754       <NA>   hg18
chrY       57772954       <NA>   hg18
chrM          16571       <NA>   hg18
```

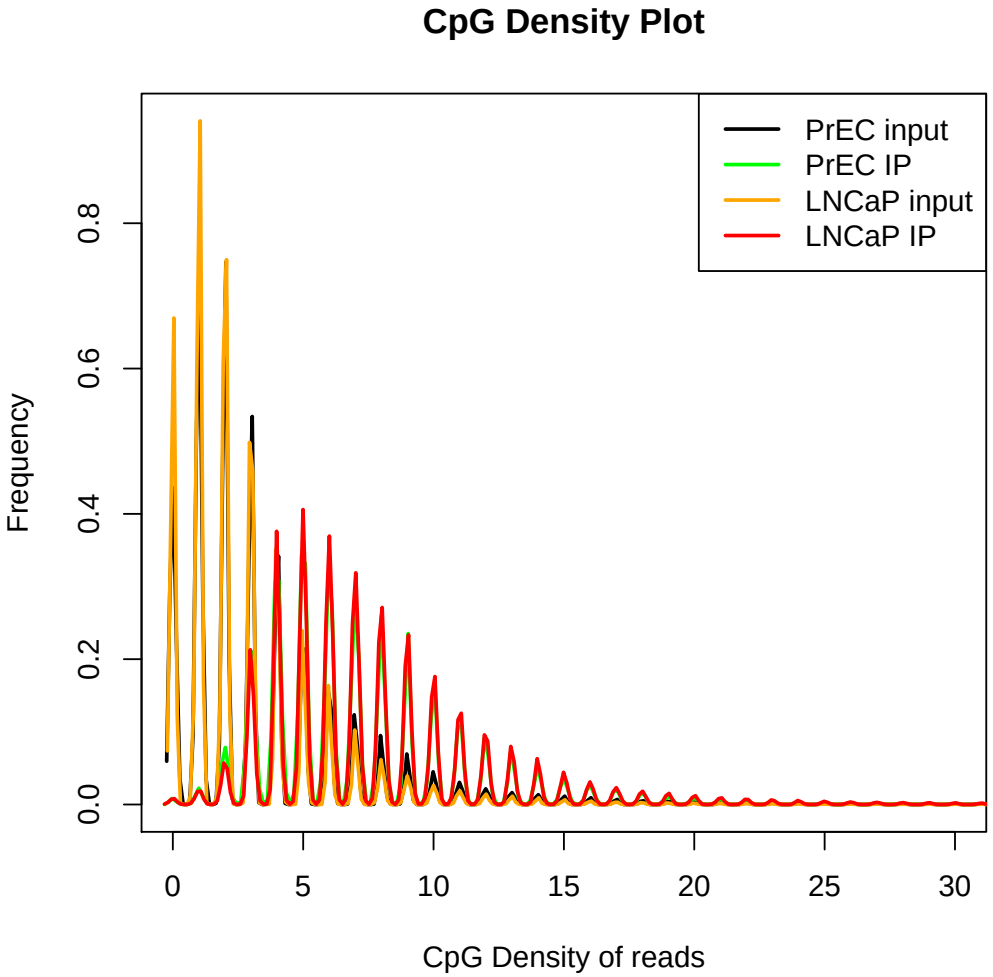
```
enrichmentPlot(samples.list, seq.len = 300,
               cols = c("black", "green", "orange", "red"),
               xlim = c(0, 10), lwd = 2)
```



The code makes use of the SeqInfo annotation of `samples.list` to retrieve the maximum base of chromosomes. Normalization scales coverage value to "reads per 10 million". The argument `seq.len=300` is passed in as the length to extend reads to, since that is approximately the real length of the fragments sequenced in this experiment. As expected, many more bases in the IP samples have high read coverages.

An alternative comparative visualization, which is somewhat specific to methylated DNA enrichment experiments, is a summary of the distribution of CpG density among reads/fragments:

```
library(BSgenome.Hsapiens.UCSC.hg18)
cpgDensityPlot(samples.list, organism = Hsapiens, w.function = "none", seq.len = 300,
               cols = c("black", "green", "orange", "red"), xlim = c(0, 30), lwd = 2)
```



The full genome sequence of the organism is required so that the (here, 300 base) DNA sequence can be fetched. In this example, the `BSgenome` package of the hg18 assembly for human is used (many other BSgenome objects for other organisms are available from Bioconductor). The `w.function` parameter allows the count of CpGs to be weighted. In this example, raw counts are used.

Notice that at lower CpG densities, the two input samples have a higher frequency of reads than the two IP samples. At higher CpG densities, this trend is reversed. This suggests that the enrichment of methylated CpGs has worked.

4 Analyses

4.1 Statistics of Differential Enrichment

The `blocksStats` function is a convenient way to do statistical tests of differential enrichment between two experimental conditions, using counts in regions of interest. The windows can be relative to some genomic landmarks, like transcription start sites (TSSs), and their size can be specified with the `up` and `down` parameters. If `up` and `down` are not provided, then windows are defined by start and end coordinates. The function leverages `edgeR`'s count modelling and its

adaptation of Fisher's exact test for assessing differential enrichment. The procedure also uses Bioconductor's facilities (i.e. `countOverlaps`) for counting mapped read in regions of the genome.

```
design.matrix <- matrix(c(0, -1, 0, 1), dimnames = list(names(samples.list), "C-N"))
design.matrix
```

```
           C-N
PrEC input   0
PrEC IP     -1
LNCaP input  0
LNCaP IP     1
```

```
stats <- blocksStats(samples.list, gene.anno, up = 2000, down = 0, seq.len = 300,
                     design = design.matrix)
```

```
Comparison of groups:  1 - -1
```

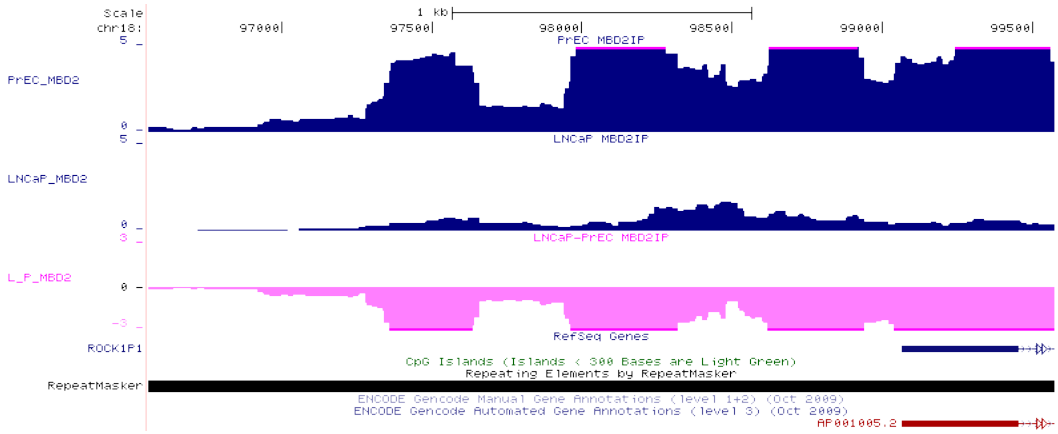
```
stats <- stats[order(stats$`adj.p.vals_C-N`), ]
head(stats)
```

```
         chr     start       end width strand    name symbol PrEC input
8019804 chr18     99064    112217 13154      + 8019804  ROCK1        600
8015798 chr17  38802738  38821439 18702      - 8015798    ---         87
7904879  chr1 145017918 145018085   168      + 7904879    ---         21
7908529  chr1 196148257 196165896 17640      + 7908529   LHX9         16
8115391  chr5 153834725 153838017  3293      - 8115391  HAND1          8
7976848 chr14 100592279 100592359    81      + 7976848    ---         29
        PrEC IP LNCaP input LNCaP IP PrEC IP_pseudo LNCaP IP_pseudo logConc_C-N
8019804     397         686       58     399.588658      57.6237948   -16.01856
8015798      56          64      314      56.365618     311.9656941   -16.21306
7904879      13          28      153      13.085296     152.0084843   -17.78513
7908529       3          13      112       3.020114     111.2740395   -19.06788
8115391       4          28       95       4.026631      94.3841477   -18.97912
7976848      75          33        1      75.489482       0.9929602   -20.14963
        logFC_C-N  p.value_C-N adj.p.vals_C-N
8019804 -2.793764 1.662518e-64   4.150642e-60
8015798  2.468516 6.449624e-44   8.051066e-40
7904879  3.538199 1.681848e-31   1.399634e-27
7908529  5.203643 1.220924e-29   7.620397e-26
8115391  4.551106 1.238883e-23   6.185989e-20
7976848 -6.247568 1.032321e-21   4.295489e-18
```

Note that this is *not* a real design matrix (in a statistical sense), it is simply a way of specifying the two experiment conditions to compare (they must be 1 and -1).

The example above calculates statistics on regions that start 2000 bases upstream of the TSS and finish at the TSS, after the reads have been extended to being 300 bases. A coverage plot from UCSC browser illustrates the best found region. For the output table, the read counts are scaled as if there were 10 million reads covering the regions of interest.

Note that this procedure only works for simple 2-group comparisons. Using this strategy for more complicated designs requires manually creating the count tables (see `annotationCounts` below) and calling the GLM-based procedures (e.g. using real design matrices) within `edgeR`.



This differential enrichment strategy can be used on bins covering the entire genome. The `genomeBlocks` function can be used to generate windows along the genome.

4.2 Domains of Concordant Change

Another analysis of interest is the detection of *regions* where changes in expression (or an epigenetic mark, etc.) occur on a particular chromosome. The function `findClusters` addresses this need. The method of determining clusters requires a search through the column of scores (e.g. t-statistics) for a persistent change. Significance of clusters is determined by randomization. The order of the statistics is permuted a large number of times and the number of clusters found in the true statistics column and the permuted statistics columns is counted, ranging from a loose cutoff to a tight cutoff. A cutoff is chosen to control the user-specified FDR. Importantly, the table must be pre-sorted in positional order. This allows the user to use whatever definition of position they want. Note that the distance between features is not taken into account in this implementation.

```
stats.table <- data.frame(chr = "chr1", pos = 1:500, t.stat = rnorm(500))
stats.table[100:104, "t.stat"] <- rnorm(5, 5)
stats.table[200:204, "t.stat"] <- rnorm(5, -5)
stats.clustered <- findClusters(stats.table, score.col = 3, w.size = 5, n.med = 2, n.con
cluster.1 <- which(stats.clustered$cluster == 1)
stats.clustered[cluster.1, ]
```

```
     chr pos      t.stat cluster
198 chr1 198 -0.56909255       1
199 chr1 199 -0.07043669       1
200 chr1 200 -5.50872460       1
201 chr1 201 -5.75598703       1
202 chr1 202 -3.41110073       1
203 chr1 203 -4.70357406       1
204 chr1 204 -7.26952975       1
205 chr1 205 -1.95311764       1
```

In this example, a running window of 5 consecutive genes is calculated along each chromosome; the median value of those 5 genes is assigned to the middle gene. If, in the 5-gene window, there are at least 2 genes that have an assigned median above the cutoff being used (cutoffs of -2, -4, -6, -8, and -10 are tried), then those genes are candidate cluster-generating genes. Starting from a candidate gene, and working outwards until encountering a positive t-statistic, if a consecutive run of at least 3 genes with t-statistic being negative could be made, then this forms a cluster. The default estimated FDR of 0.05 is used.

4.3 Finding enriched regions in a single sample

Repitools contains an implementation of `ChromaBlocks` (described in Hawkins RD et al), designed to discover regions of the genome which are enriched for epigenetic marks, such as H3K27me3. Briefly, ChromaBlocks counts the number of sequencing reads aligned to adjacent bins in the genome in both Immunoprecipitated and Input samples, determines which bins exceed a cutoff for IP-Input enrichment (either specified or set at a supplied FDR by permutation) and returns regions of the genome where multiple adjacent bins are enriched.
Data from the Human Reference Epigenome Mapping Project is used to demonstrate `ChromaBlocks`. The data was downloaded from here. Samples GSM466734, GSM466737, GSM466739, and GSM450270 are used here. The file containing the reads is available for download from the University of Zurich. After loading it into your R session, run the commands:

```
 class(H1samples)

[1] "GRangesList"
attr(,"package")
[1] "GenomicRanges"

 names(H1samples)

[1] "H3K4me1"  "H3K27me3" "H3K36me3" "Input"

 S4Vectors::elementNROWS(H1samples)

 H3K4me1 H3K27me3 H3K36me3    Input
 1201402  8673675  4151895 15289957

 H3K27me3.blocks <- ChromaBlocks(rs.ip = H1samples["H3K27me3"],
                                 rs.input = H1samples["Input"],
                                 organism = Hsapiens, chrs = "chr20",
                                 preset = "small", seq.len = 300)

Permutation 1.
Permutation 2.
Permutation 3.
Permutation 4.
Permutation 5.
Testing positive regions.
Using cutoff of 2.337163 for a FDR of 0.01
.
```

`ChromaBlocks` returns a `ChromaResults` object, from which an `IRangesList` of the regions determined to be enriched can be retrieved using the `regions` method.

```
 H3K27me3.blocks
```

```
Object of class 'ChromaResults'.
1400 regions found with using a cutoff of 2.337163
```

```
 regions(H3K27me3.blocks)
```

```
IRangesList of length 1
$chr20
IRanges of length 1400
          start      end width
[1]       60150    61550  1401
[2]      189750   191050  1301
[3]      228950   236350  7401
[4]      237050   243050  6001
[5]      243450   245550  2101
[6]      245850   248250  2401
[7]      259250   260950  1701
[8]      275950   277950  2001
[9]      279350   281050  1701
...         ...      ...   ...
[1392] 62185050 62186750  1701
[1393] 62204850 62206750  1901
[1394] 62272950 62274750  1801
[1395] 62282050 62285250  3201
[1396] 62362950 62364450  1501
[1397] 62367750 62371150  3401
[1398] 62404450 62405650  1201
[1399] 62407650 62409450  1801
[1400] 62426850 62428550  1701
```

5 Visualisations

The visualisations in the following two subsections manipulate matrices of scores (such as coverages or intensities) in some way, such as clustering them, or summarising them by some defined grouping. The common interface for creating matrices of scores at regular distances from a reference point, such a TSS, is the `featureScores` function. The following example samples smoothed coverages between 5000 bases upstream of gene TSSs, and 1000 bases downstream of the TSS, at 1000 base intervals, in the PrEC and LNCaP immunoprecipitated samples. The scores are then subtracted from each other.

```
 prostateCvgs <- featureScores(samples.list[c("PrEC IP", "LNCaP IP")], gene.anno,
                               up = 5000, down = 1000, freq = 1000, s.width = 500)
 prostateCvgs
```

```
An object of class 'ScoresList'.
Tables: PrEC IP, LNCaP IP.
Features:
GRanges with 24966 ranges and 2 elementMetadata values:
          seqnames               ranges strand   |      name      symbol
             <Rle>            <IRanges>  <Rle>   | <integer> <character>
      [1]     chr1   [ 781253,  783614]      +   |   7896759   LOC643837
      [2]     chr1   [ 850983,  869824]      +   |   7896761      SAMD11
      [3]     chr1   [ 885829,  890958]      +   |   7896779      KLHL17
      [4]     chr1   [ 891739,  900345]      +   |   7896798     PLEKHN1
      [5]     chr1   [ 938709,  939782]      +   |   7896817       ISG15
      [6]     chr1   [ 945365,  981355]      +   |   7896822        AGRN
      [7]     chr1   [1092346, 1092441]      +   |   7896859         ---
      [8]     chr1   [1093105, 1093195]      +   |   7896861         ---
      [9]     chr1   [1094247, 1094330]      +   |   7896863         ---
      ...      ...                 ...    ... ...       ...         ...
  [24958]     chrY [19611913, 19614093]      -   |   8177222        CD24
  [24959]     chrY [19640262, 19640361]      -   |   8177227         ---
  [24960]     chrY [20076704, 20124427]      -   |   8177229      BCORL2
  [24961]     chrY [20326698, 20366197]      -   |   8177232     JARID1D
  [24962]     chrY [21036941, 21090502]      -   |   8177261      TTTY10
  [24963]     chrY [21954227, 21957634]      -   |   8177273         ---
  [24964]     chrY [22001970, 22008519]      -   |   8177280         ---
  [24965]     chrY [22154873, 22165940]      -   |   8177282      TTTY13
  [24966]     chrY [22852332, 22854411]      -   |   8177344       TTTY5
  ---
  seqlengths:
    chr1 chr10 chr11 chr12 chr13 chr14 ...  chr6  chr7  chr8  chr9  chrX  chrY
      NA    NA    NA    NA    NA    NA ...    NA    NA    NA    NA    NA    NA
Region: 5000 bases up to 1000 bases down.
Smoothing: 500, 500 bases.
Sampling : 1000 bases.
```

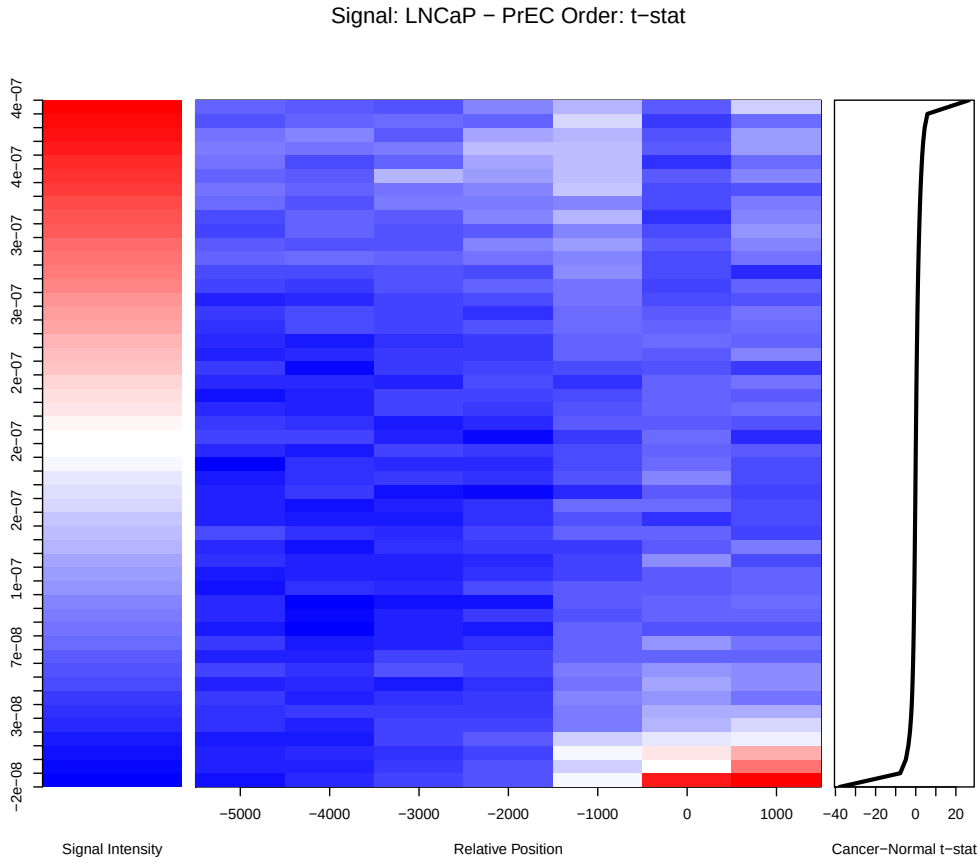
```
prostateCvgs@scores <- list(tables(prostateCvgs)[[2]] - tables(prostateCvgs)[[1]])
names(prostateCvgs) <- "LNCaP - PrEC"
```

This object will be used in the next few visualisation functions.

5.1 Integrative analysis of epigenetics and gene expression

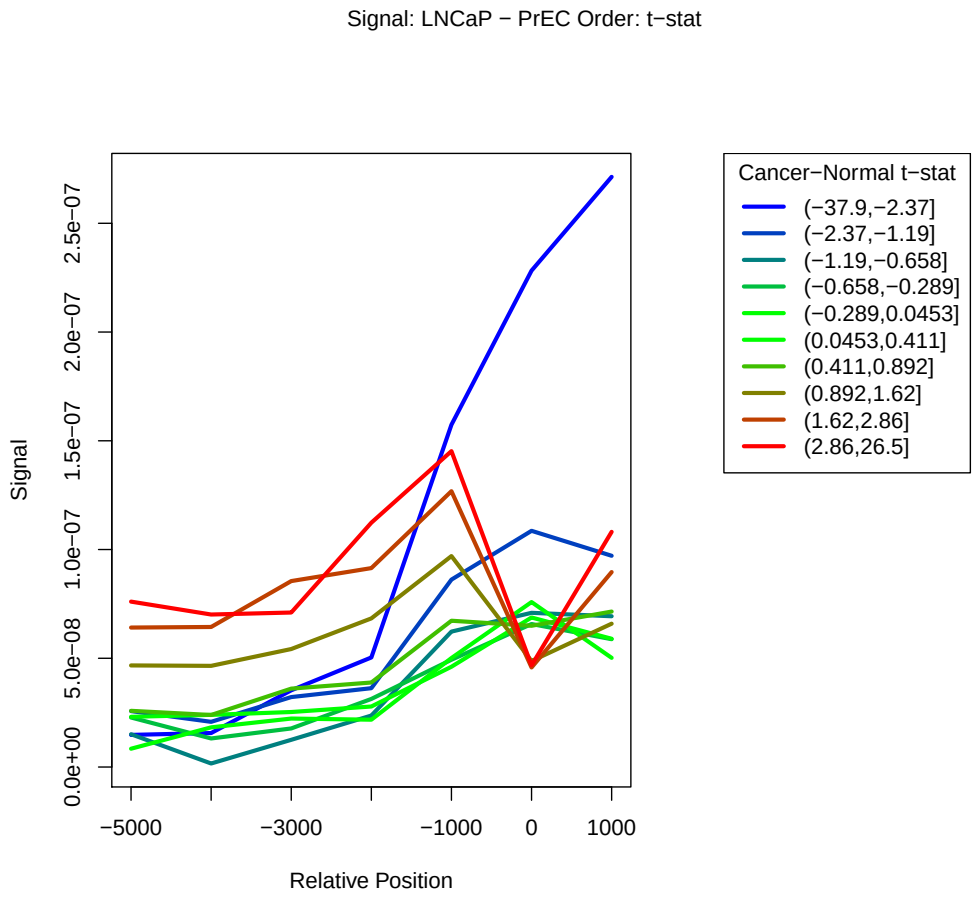
Epigenomic data is often gathered with other data, such as gene expression. It may be of interest to see the profile of epigenetic enrichment at a variety of distances from TSSs, stratified by gene expression level. The `binPlots` function is a convenient way to look at these interactions.

```
binPlots(prostateCvgs, ordering = expr, ord.label = "Cancer-Normal t-stat",
         plot.type = "heatmap", n.bins = 50)
```



Enrichment levels (here, differential enrichment) are split into bins based on the moderated t-statistics for change in expression. Signal for (differential) enrichment is averaged over genes in the bin and plotted as a heatmap. As expected, the genes that are silenced in cancer exhibit higher levels of DNA methylation around their TSS, compared to normal cells. This visualization can be represented as a lineplot, by setting `plot.type= "line"` (see below).

```
binPlots(prostateCvgs, ordering = expr, ord.label = "Cancer-Normal t-stat",
        plot.type = "line", n.bins = 10)
```

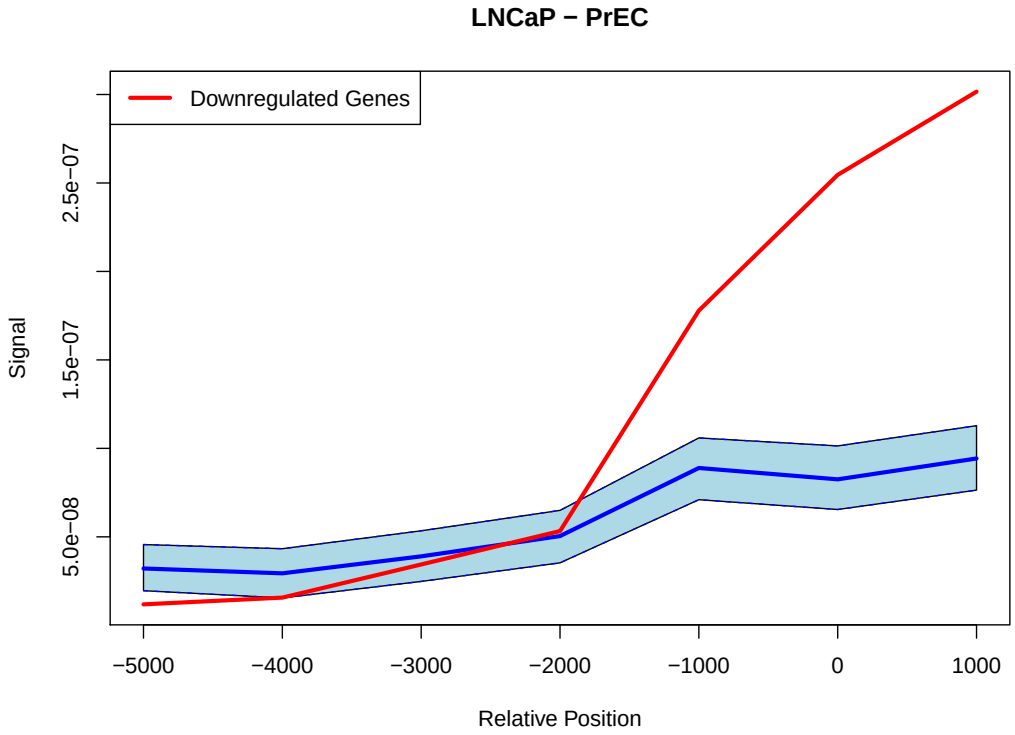


This strategy is useful for determining the location (e.g. relative to TSS) signal most often occurs relative to expression and can be coupled to ranked gene expression levels, instead of differential expression. These determined regions of interest relative to TSS can then be used in targeted analyses (e.g. `blocksStats`, see above).

5.2 Gene Set Enrichment

Sets of genes (e.g. genes disrupted in a certain type of cancer, or differentially expressed between experimental conditions) are ever-present in genomics research. For such genes of interest, the profile of intensities or counts can be plotted versus the profile of randomly selected gene lists using the `profilePlots` function. In the following example, the DNA methylation profile of genes silenced in cancer is significantly different to random sets of genes.

```
which.loss <- which(expr < -3)
profilePlots(prostateCvgs,
             gene.lists = list(`Downregulated Genes` = which.loss),
             cols = "red")
```



The blue region forms the "null" distribution that was created by sampling random gene lists of the same size as the user-specified gene list a number of times, as set by the `nSamples` parameter. By default, the null region is a between the 0.025 and 0.975 quantiles of the null distribution. In this example, it appears that the genes silenced in cancer have a significant gain of methylation 2000 bases either side of the TSSs, in comparison to random sets of other genes.

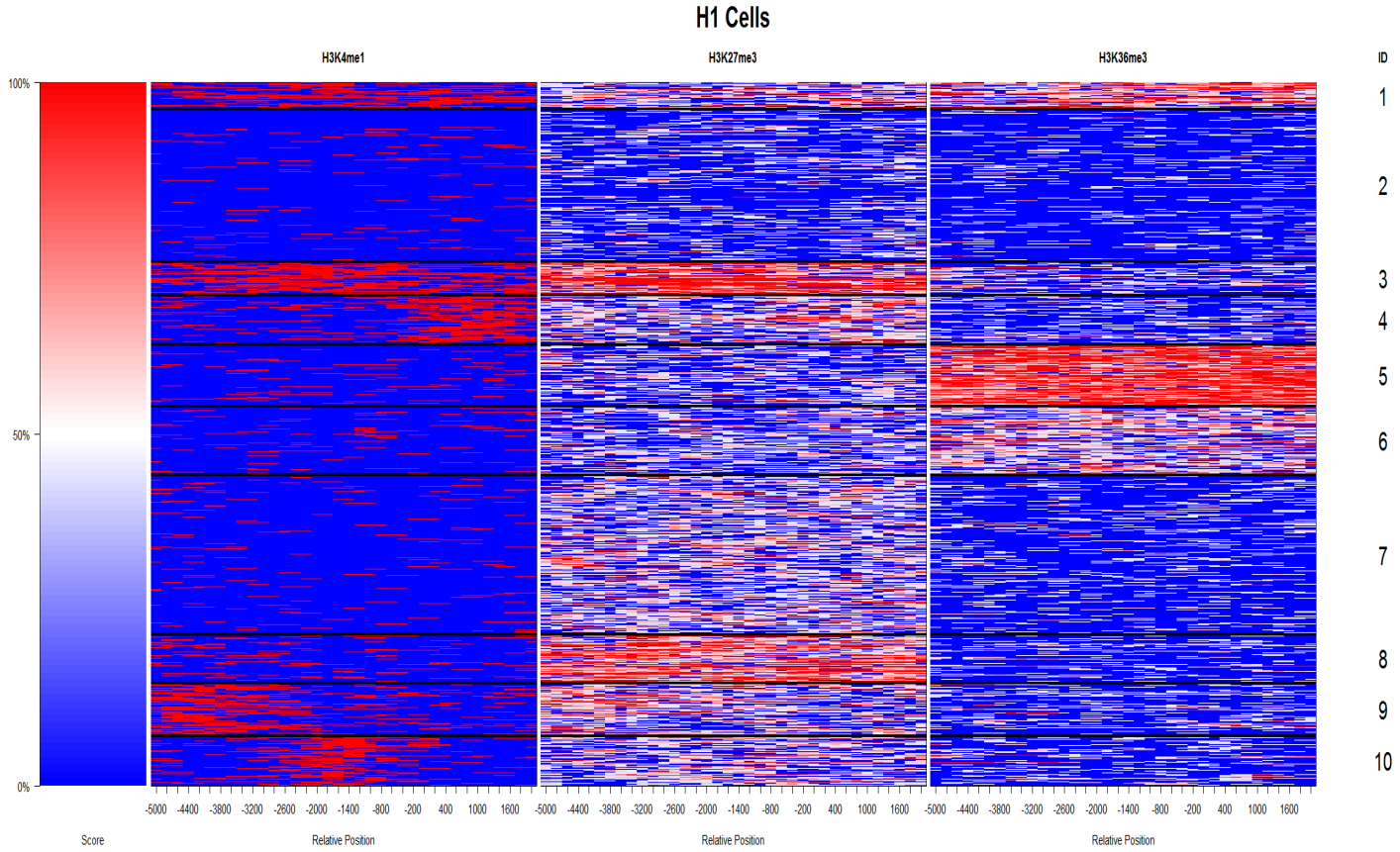
5.3 Clustering epigenomic signals

`clusterPlots` is another way to look at read depth at regular positions around a feature (e.g. TSS). The first step is to use `featureScores` to get the coverage tables, which essentially gives a list of coverage tables for the samples used. `clusterPlots` is then called, which does k-means clustering, or if the user wants to use their own clustering algorithm, the cluster ID of each feature can be passed in. In any case, the features are grouped by their cluster memberships and plotted as either a heatmap with one row for every feature, or a set of lineplots showing the average coverage of all features belonging to each cluster. If gene expression data is also available, it can be plotted alongside the heatmaps.

```
cvgs <- featureScores(H1samples[1:3], gene.anno, up = 5000, down = 2000, dist = "base",
                      freq = 200, s.width = 500)

cp <- clusterPlots(cvgs, scale = function(x) sqrt(x), plot.type = "heatmap",
             t.name = "H1 Cells", n.clusters = 10)
```

```
null device
          1
```



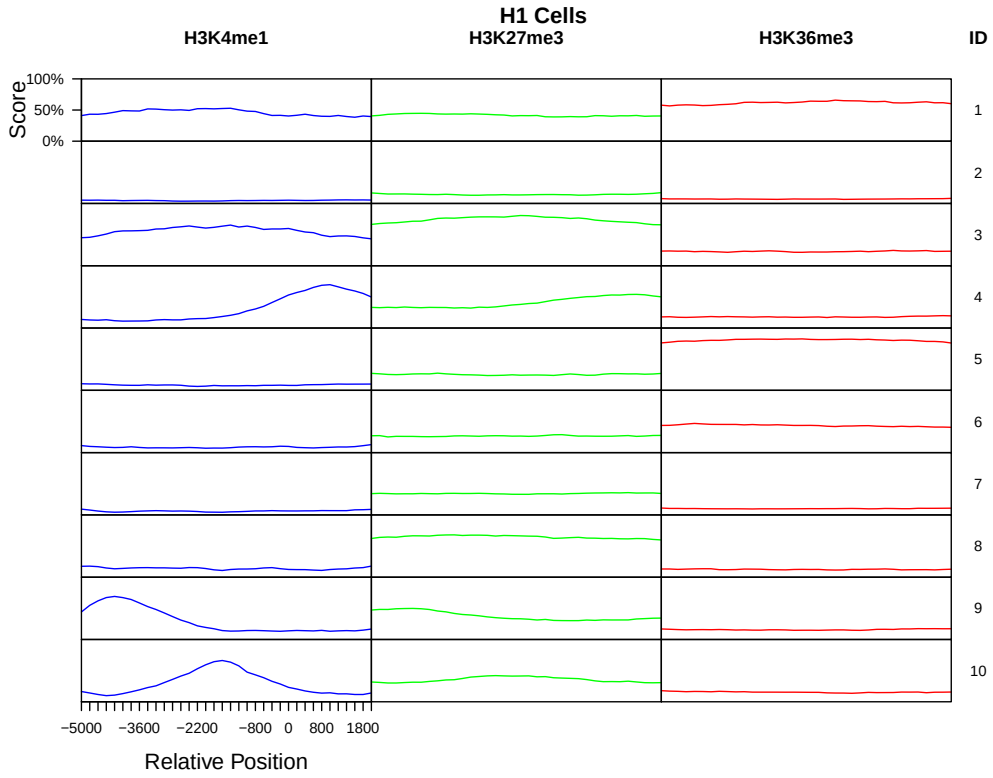
Here, we have scaled the signal using the square root transformation. If you don't specify this, no scaling is done.

Note that we have saved the output of `clusterPlots` (a `ClusteredCoverageList` object), which can be plotted in alternative ways, such as line plots:

```
table(clusters(cp))
```

```
   1    2    3    4    5    6    7    8    9   10
 950 5399 1187 1770 2175 2450 5675 1698 1892 1770
```

```
clusterPlots(cp, plot.type = "line", t.name = "H1 Cells")
```



Also, this allows users to do their own clustering and use `clusterPlots` for the plotting, or to extract the cluster identifiers for downstream analyses (e.g. functional category analysis). Furthermore, in addition to specifying a vector of expression values and plotting it alongside the clustered epigenetic signal, users can give an additional vector in the `sort.data` argument to sort on within a cluster (e.g. gene length, CpG density, etc.).

6 Utility Functions

The function described in this section perform useful tasks that are commonly made with epigenetic data.

6.1 Windows and Counts

Often, it is required to create a set of windows covering the entire genome, for some analysis. The function `genomeBlocks` does this.

```
chrs <- c(50000, 10000)
names(chrs) <- c("chr1", "chr2")
genomeBlocks(chrs, width = 5000)
```

```
GRanges object with 12 ranges and 0 metadata columns:
       seqnames      ranges strand
          <Rle>   <IRanges>  <Rle>
   [1]     chr1       1-5000      *
   [2]     chr1  5001-10000      *
   [3]     chr1 10001-15000      *
   [4]     chr1 15001-20000      *
```

```
  [5]     chr1 20001-25000      *
  ...      ...         ...    ...
  [8]     chr1 35001-40000      *
  [9]     chr1 40001-45000      *
 [10]     chr1 45001-50000      *
 [11]     chr2      1-5000      *
 [12]     chr2  5001-10000      *
 -------
 seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

This example makes a `GRanges` object of 5 kb windows along both example chromosomes.

`annotationCounts` is useful to tally the counts of reads surrounding some set of genomic landmarks. `annotationBlocksCounts` is the analogous function for counting in user-specified regions of the genome.

```
annotationCounts(samples.list, head(gene.anno, 10), up = 2000, down = 500,
                 seq.len = 300)
```

```
        PrEC input PrEC IP LNCaP input LNCaP IP
7896759         25      35          29       69
7896761         10       2           8       36
7896779         11      15          10       14
7896798         19      61          15       83
7896817         20      41          22       46
7896822         11      17           8       28
7896859         24      35           8       70
7896861         21      57           6       78
7896863         18      85           8       85
7896865         22      92          18       90
```

This example counts reads that fall within 2000 bases upstream and 500 bases downstream of (the first ten) TSSs in the gene annotation table. Reads are extended to 300 bases.

6.2 Characteristics of the DNA sequence

It would be useful to know when seeing a lack of reads in some windows, if the mappability of the window is the cause. Some regions of the genome have low complexity sequence, where reads are unlikely to map uniquely to. The function `mappabilityCalc` calculates the percentage of each region that can be mapped to by reads generated from the experiment. It operates on a user-created `BSgenome` object of a masked genome sequence. The definition of which bases are mappable and which are not depends on the read length of the sequencing technology used. Therefore, there is no one masked `BSgenome` object that can be used by all users. Note that by masking, we mean replacing the unmappable reference sequence bases by 'N', not creating a built-in mask.

```
library(BSgenome.Hsapiens36bp.UCSC.hg18mappability)
locations <- data.frame(chr = c("chr4", "chr9"),
                        position = c(50000000, 100000000))
mappabilityCalc(locations, organism = Hsapiens36bp, window = 500, type = "center")
```

```
 <NA>  <NA>
0.000 0.998
```

The region on chromosome 4 is completely unmappable, whereas the region on chromosome 9 is almost completely mappable.

Next, we may be interested in determining CpG density of a region.

```
 cpgDensityCalc(head(gene.anno, 10), window = 100, organism = Hsapiens)
```

```
 [1]  0 10 16  7 10 20  6  4  6  4
```

This example calculates the CpG density of a window 100 bases either side of the TSS for the first ten genes in the gene annotation table. By default, the CpG density is just the raw number of counts in the windows. There are also linearly, exponentially and logarithmically decaying weight schemes available.

7 Summary

Repitools has a number of useful functions for quality checking, analysis, and comparison of trends. Many of the functions work seamlessly on array data, as well as sequencing data. Also, there are numerous utility functions, that perform some common task in the investigation of epigenomic data. Consult the package documentation for instructions on how to use functions that were not demonstrated by this vignette.

8 Environment

This vignette was created in:

```
 sessionInfo()
```

```
R version 2.14.1 (2011-12-22)
Platform: x86_64-pc-mingw32/x64 (64-bit)

locale:
[1] LC_COLLATE=English_Australia.1252  LC_CTYPE=English_Australia.1252
[3] LC_MONETARY=English_Australia.1252 LC_NUMERIC=C
[5] LC_TIME=English_Australia.1252

attached base packages:
[1] grid      stats     graphics  grDevices utils     datasets  methods
[8] base

other attached packages:
 [1] BSgenome.Hsapiens36bp.UCSC.hg18mappability_1.0
 [2] cluster_1.14.1
```

```
 [3] gplots_2.10.1
 [4] KernSmooth_2.23-7
 [5] caTools_1.12
 [6] bitops_1.0-4.1
 [7] gdata_2.8.2
 [8] gtools_2.6.2
 [9] Ringo_1.18.0
[10] Matrix_1.0-2
[11] lattice_0.20-0
[12] limma_3.10.2
[13] RColorBrewer_1.0-5
[14] Biobase_2.14.0
[15] edgeR_2.0.5
[16] BSgenome.Hsapiens.UCSC.hg18_1.3.17
[17] BSgenome_1.22.0
[18] Biostrings_2.22.0
[19] GenomicRanges_1.6.6
[20] IRanges_1.12.5
[21] Repitools_1.0.11

loaded via a namespace (and not attached):
[1] annotate_1.32.0      AnnotationDbi_1.16.0 DBI_0.2-5
[4] genefilter_1.36.0    RSQLite_0.10.0       splines_2.14.1
[7] survival_2.36-10     tools_2.14.1         xtable_1.6-0
```