Performing Some Basic Quality Checking and Analysis on Sequencing Data with `Repitools`

Mark Robinson Aaron Statham Dario Strbenac

Last compiled on: April 20, 2011

1 Introduction

`Repitools` is a package that allows statistics of absolute or differential binding for ChIP-seq and MeDIP-seq to be calculated, as well as summaries of genome-wide trends to be visualised in a variety of formats. Some basic quality checking utilities are also available for sequencing data. Most of the functionality available is implemented for both microarrays and next generation sequencing, with very similar function calls for both types of data.

In this vignette, quality checks of the sequencing data, followed by analysis and visualisation will be demonstrated. Further description of the package can be found in the associated Bioinformatics Applications Note [1]

To start with, load the `Repitools` package.

```
library(Repitools)
```

2 Example Datasets

A `GRangesList` of mapped short reads from an Illumina Genome Analyser run of four samples is included with the package. This data has been published and is available here. LNCaP is the cancer cell line, and PrEC is the normal cell line. `GRanges` objects of mapped files from many popular aligners can be created by first reading them into **R** with the `readAligned` function in the `ShortRead` package, then coerced with `as(alnRdObj, "GRanges")`. The two convenience methods `BAM2GRanges` and `BAM2GRangesList` in `Repitools` could also be used, if the reads were stored on disk in BAM format. 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 **R**.

```
library(GenomicRanges)
load("samplesList.RData")
class(samples.list)
```

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

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

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

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

```
elementLengths(samples.list)
```

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

Also, an annotation of genes will be used. The annotation is based on one provided from Affymetrix with their expression arrays [2]. This is to be able to relate the sequencing data gene expression measurements done on an array.

```
gene.anno <- read.csv("geneAnno.csv", 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
```

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

```
load("expr.RData")
head(expr)
```

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

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

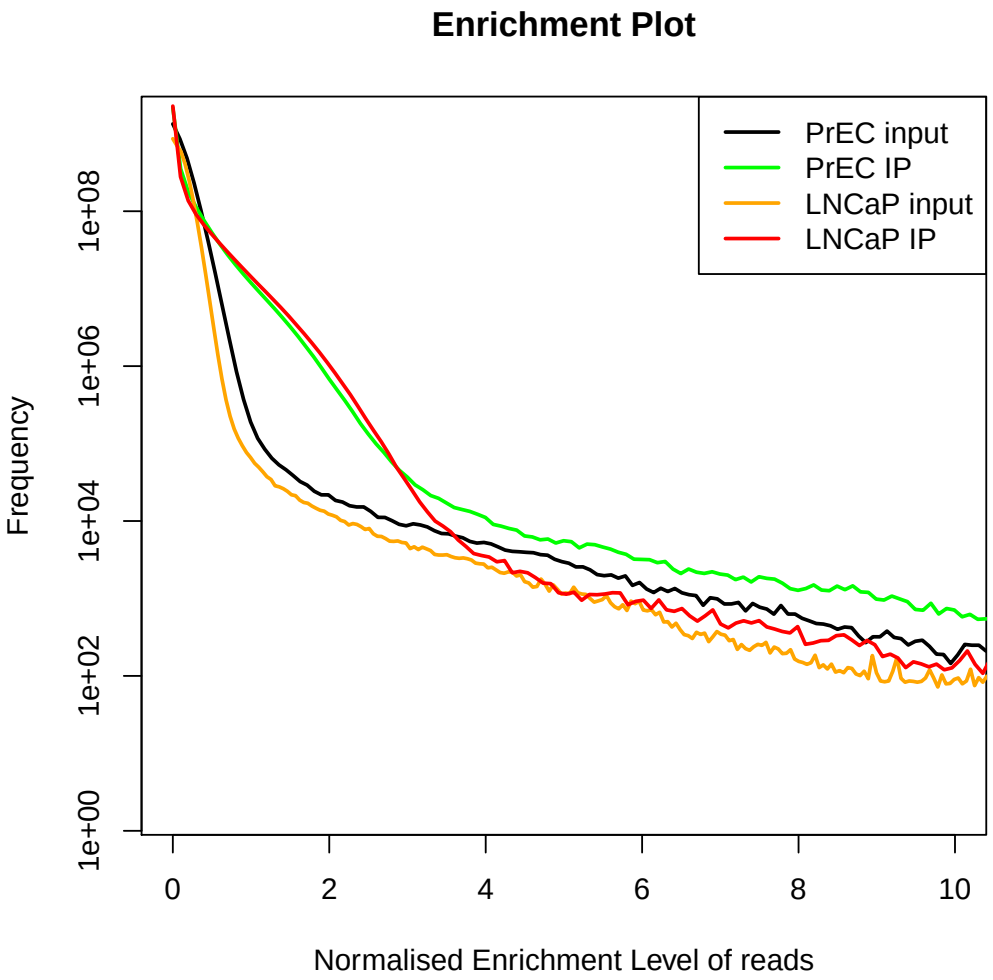
3 Quality Checking

Notice that two of the samples are MBD2 IPs, and two are inputs. Therefore, the IP samples should differ to the inputs in two ways. Firstly, they should be more CpG rich, since DNA methylation rarely ever occurs outside of this sequence context. Also, since DNA methylation tends to occur in peaks, rather than spread out regions, a higher frequency of bases should have high coverage of reads in the IP samples than in input samples. The `enrichmentPlot` and `cpgDensityPlot` functions allow examination of this.

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

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

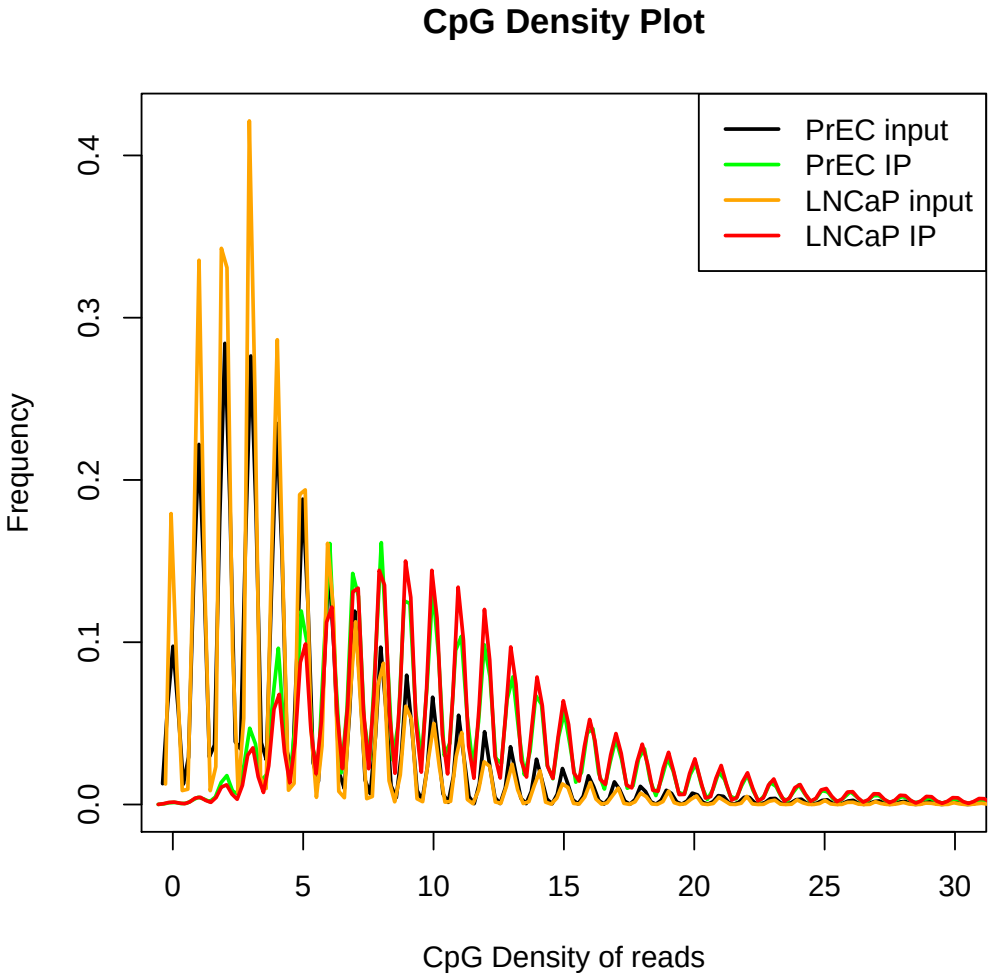
```
enrichmentPlot(samples.list, seq.len = 300, cols = c("black",
    "green", "orange", "red"), xlim = c(0, 10), lwd = 2)
```



The above code uses the SeqInfo annotation of samples.list to get the maximum base of chromosomes. The normalisation of the coverage used is to scale every coverage value by 10 $million/number\_of\_reads\_$ 300 is passed in as the `seq.len` parameter, because that is approximately the real length of the fragments sequenced. As expected, many more bases in the IP samples have high read coverages.

Next, the CpG density of reads is examined.

```
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 being analysed is required so that the 300 base DNA sequence (tags are only 36 bp long) may be fetched. In this example, the `BSgenome` package of the hg18 assembly for human is used. There are many BSgenome objects for other organisms available. See the Bioconductor website for more information. 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 DNA methylation IP has worked.

4 Analyses

4.1 Statistics of Differential Enrichment

The `blocksStats` function is a convenient way to do a statistical test of differential enrichment between two groups or treatments, for counts in windows. The windows can be relative to some genomic landmarks, like TSSs, and their size can be specified with the `up` and `down` parameters. If `up` and `down` are not provided, then the windows defined by that start and end coordinates of

the annotation are used. The function leverages `edgeR`'s count modelling and its adaptation of Fisher's exact test for assessing differential enrichment.

```
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
```

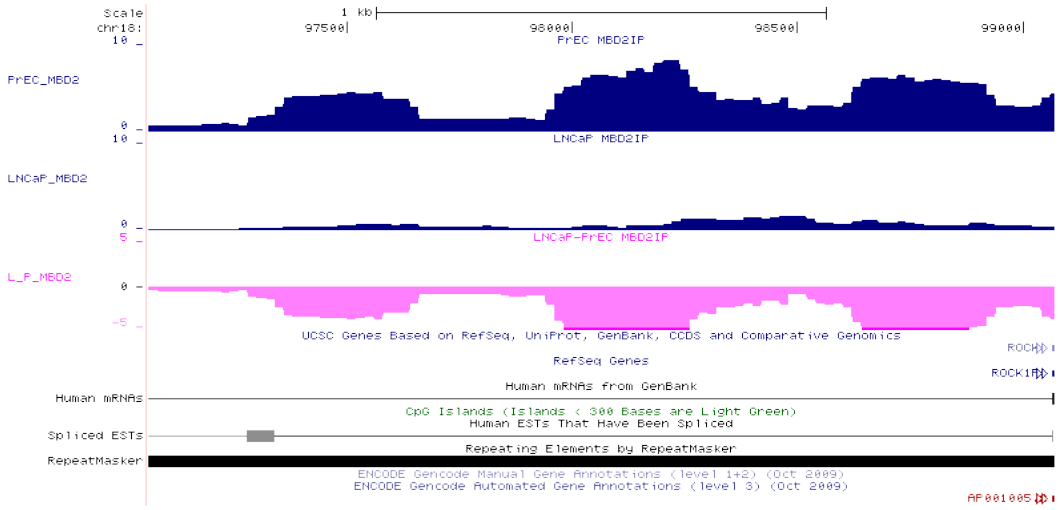
```
stats <- blocksStats(samples.list, gene.anno, up = 2000, down = 0,
    seq.len = 300, design = design.matrix)
```

```
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
7976073 chr14  85066240  85164023 97784      + 7976073  FLRT2         17
        PrEC IP LNCaP input LNCaP IP PrEC IP_pseudo LNCaP IP_pseudo logConc_C-N
8019804     397         686       58   3.995887e+02        57.62379   -16.01856
8015798      56          64      314   5.636562e+01       311.96569   -16.21306
7904879      13          28      153   1.308530e+01       152.00848   -17.78513
7908529       3          13      112   3.020114e+00       111.27404   -19.06788
8115391       4          28       95   4.026631e+00        94.38415   -18.97912
7976073       0          20       69   1.435022e-11        68.55255   -33.59047
        logFC_C-N  p.value_C-N adj.p.vals_C-N
8019804 -2.793764 9.642793e-36   2.407420e-31
8015798  2.468516 3.662695e-24   4.572142e-20
7904879  3.538199 3.672510e-18   3.056262e-14
7908529  5.203643 7.401219e-18   4.619471e-14
8115391  4.551106 2.875722e-14   1.435905e-10
7976073 32.851160 1.911561e-13   7.954003e-10
```

The example calculates statistics on TSS regions which start 2000 bases upstream of the TSS and finish at the TSS, after the reads have been extended to being 300 bases long. A coverage plot from UCSC browser illustrates the best found region. By default, the coverage values are scaled to be as if there were 10 million reads in each lane.



4.2 Domains of Concordant Change

Another analysis of interest for the epigenomics research community is to find regions of the genome where epigenetic marks or changes in such marks occur in consecutive genes on a particular chromosome. The function `findClusters` addresses this need. The method of determining clusters is to look through the column of scores for a set of consecutive scores in the same direction. Which potential clusters are significant is determined by randomising the ordering of the statistics column a number of times, and counting the number of clusters found in the real statistics column and the randomised statistics columns, from a loose cutoff to a tight cutoff, and choosing the cutoff to be the first cutoff that meets or is below 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 and sort by that definition.

```
stats.table <- cbind(gene.anno, expr)
stats.table$pos <- ifelse(stats.table$strand == "+", stats.table$start,
    stats.table$end)
pos.order <- order(stats.table$chr, stats.table$pos)
stats.table <- stats.table[pos.order, ]
stats.clustered <- findClusters(stats.table, score.col = 7, w.size = 5,
    n.med = 2, n.consec = 3, cut.samps = seq(-2, -10, -2), maxFDR = 0.05,
    trend = "down", n.perm = 10)
cluster.1 <- which(stats.clustered$cluster == 1)
stats.clustered[cluster.1, ]
```

```
          name  chr strand    start      end  symbol      t-stat      pos
7914667 7914667 chr1      - 33829993 33947691   CSMD2  -0.8496609 33947691
7899898 7899898 chr1      + 34102217 34102979   HMGB4  -0.2014024 34102217
7899905 7899905 chr1      + 34102217 34102979   HMGB4  -0.1829972 34102217
7914748 7914748 chr1      - 34251727 34252405     ---  -0.3865665 34252405
7899911 7899911 chr1      + 34405070 34457319 C1orf94  -0.9322332 34405070
7899921 7899921 chr1      + 34993307 34996699    GJB5 -16.4867896 34993307
7899927 7899927 chr1      + 34999364 35000515    GJB4  -8.2965082 34999364
7899932 7899932 chr1      + 35019376 35024552    GJB3 -11.7589771 35019376
7899939 7899939 chr1      + 35031185 35033935    GJA4  -0.2142715 35031185
        cluster
```

```
7914667       1
7899898       1
7899905       1
7914748       1
7899911       1
7899921       1
7899927       1
7899932       1
7899939       1
```

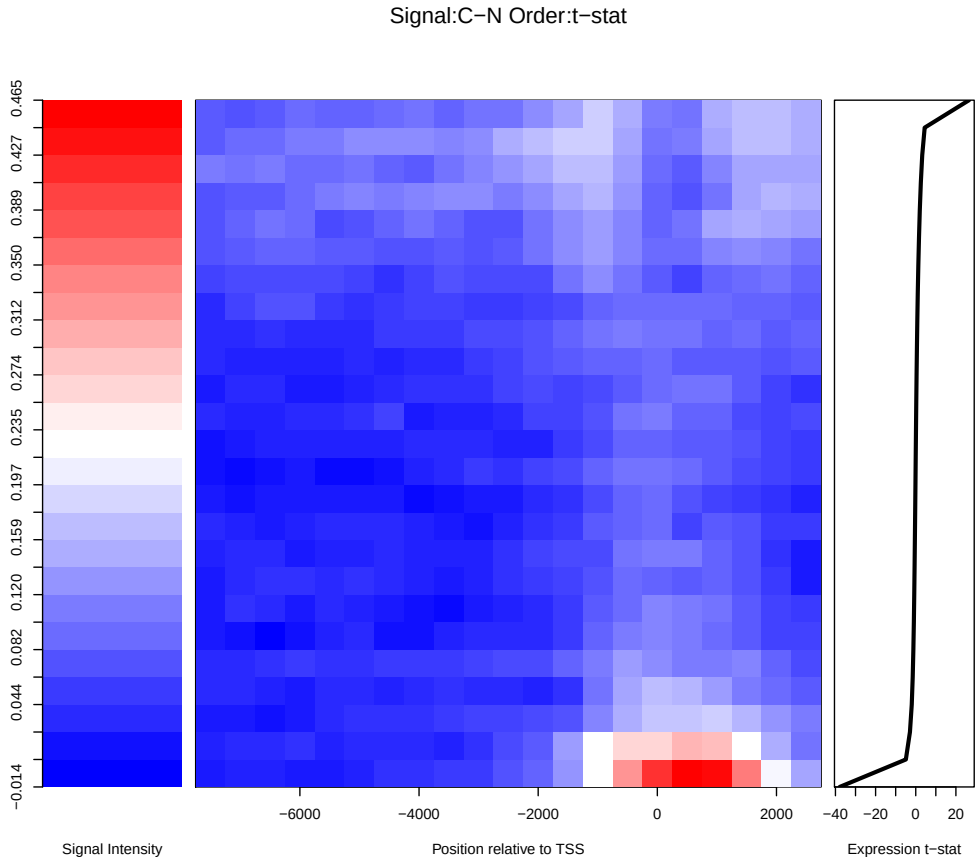
In this example, a running window of 5 consecutive genes is run across every chromosome, and the median value of those 5 genes is assigned to the middle gene. If, in the 5-gene window, there are at least 2 genes that have an assigned median above the cutoff being used (cutoffs of -2, -4, -6, -8, and -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 FDR of 0.05 is used.

5 Visualisations

5.1 Relating Epigenetics and Expression

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

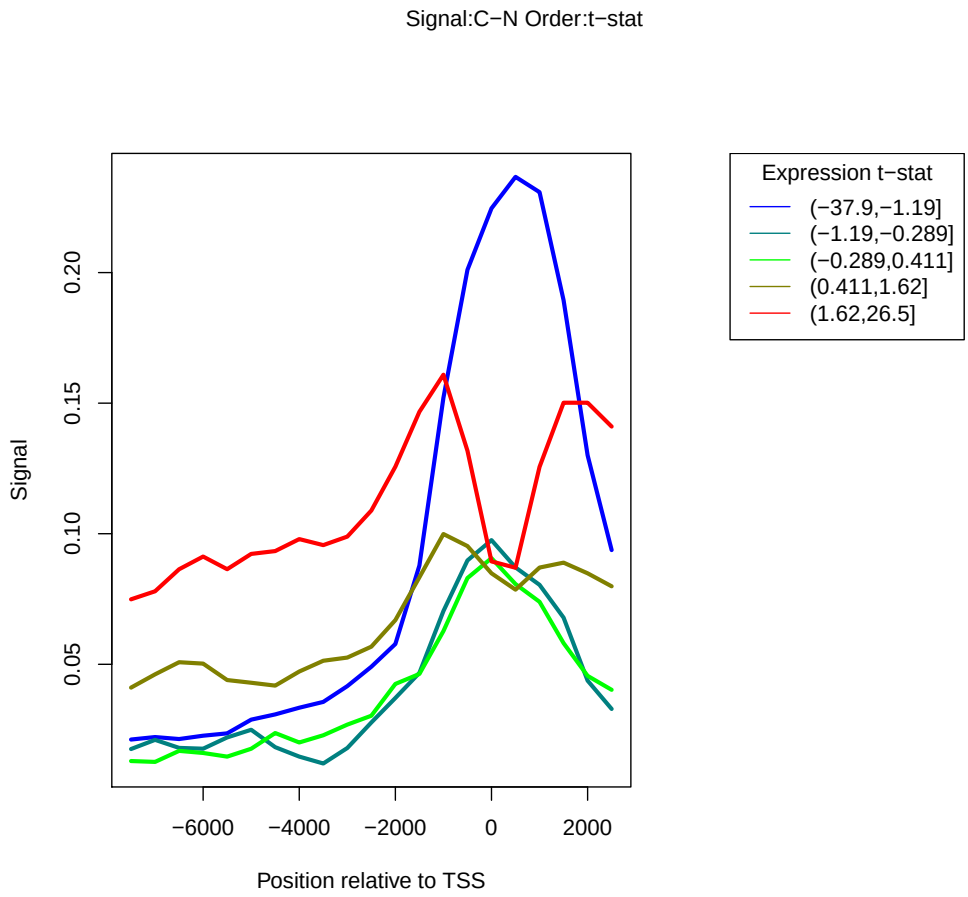
```
binPlots(samples.list, gene.anno, design = design.matrix, up = 7500,
    down = 2500, by = 500, bw = 500, seq.len = 300, ordering = expr,
    ordLabel = "Expression t-stat", plotType = "heatmap", nbins = 25)
```



This example made counts in 500 base non - overlapping windows between -7500 bases upstream and 2500 bases downstream (the default range) for each gene, then split them into categories based on the expression difference value, and averaged over all counts for each particular window and expression category. It is encouraging to see that the lowest level of expression has a rather fine enrichment of DNA methylation about 2000 bases either side of the TSS. Apart from the heatmap visualisation, there are a number of other styles. Details can be found in the documentation of the function.

To demonstrate how similar it is to generate another style of binned plot, the next example shows the same data as a line-plot. Note that the function call is the same, apart from the `plotType` parameter. Notice the spike in DNA methylation for the set of lowest expressed genes, which form the blue line.

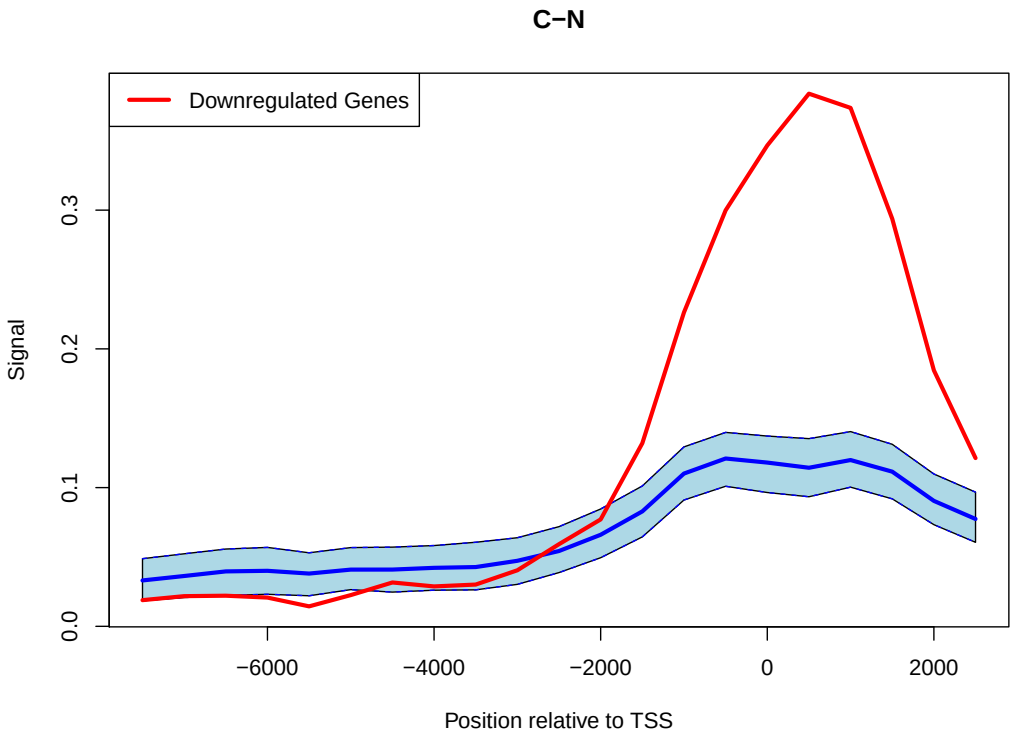
```
binPlots(samples.list, gene.anno, design = design.matrix, up = 7500,
    down = 2500, by = 500, bw = 500, seq.len = 300, ordering = expr,
    ordLabel = "Expression t-stat", plotType = "line", nbins = 5)
```



5.2 Gene Set Enrichment

Some genes may be of interest to the researcher for some reason. This subset of genes may be known to be strongly marked with another epigenetic mark, or change in expression in the same direction strongly, or many other reasons. No matter what the reason for selecting the subset is, the profile of intensities or counts can be plotted versus the profile of randomly selected gene lists and compared with the `significancePlots` function. In the following example, it will be checked whether the DNA methylation profile of genes losing expression is significantly different to random gene sets.

```
which.loss <- which(expr < -3)
significancePlots(samples.list, anno = gene.anno, up = 7500,
    down = 2500, geneList = list(`Downregulated Genes` = which.loss),
    design = design.matrix, by = 500, bw = 500, seq.len = 300)
```



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.

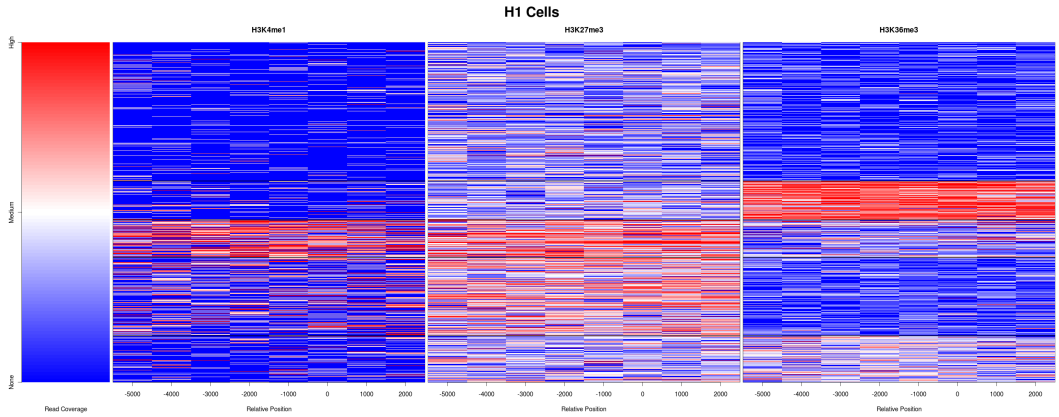
clusterPlots is another way to look at read depth at regular positions around a feature. The first step is to use `featureScores` to get the coverage tables, which essentially gives a list of coverage tables for the samples used. `clusterPlots` is then called, which does the simple and fast k-means clustering, 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.

Data from the Human Reference Epigenome Mapping Project is used to demonstrate this visualisation. The data was downloaded from here. Samples GSM466734, GSM466737, and GSM466739 are used.

```
load("H1samples.RData")
class(H1samples)
```

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

```
names(H1samples)
```




```
[1] "H3K4me1"  "H3K27me3" "H3K36me3"
```

```
elementLengths(H1samples)
```

```
H3K4me1 H3K27me3 H3K36me3
1201402  8673675  4151895
```

```
cvgs <- featureScores(H1samples, gene.anno, up = 5000, down = 2000,
    dist = "base", freq = 1000, s.width = 1000)
```

```
clusterPlots(cvgs, function(x) sqrt(x), plot.type = "heatmap",
    t.name = "H1 Cells")
```

```
pdf
  2
```

It appears that high levels of H3K36me3 are associated with low levels of H3K4me1.

6 Utility Functions

These functions perform some task that is commonly made with the data, but is not a formal analysis.

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.

```
genomeBlocks(Hsapiens, chrs = 1:25, width = 5000)
```

```
GRanges with 616087 ranges and 0 elementMetadata values
         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]      *   |
     [6]     chr1       [25001, 30000]      *   |
     [7]     chr1       [30001, 35000]      *   |
     [8]     chr1       [35001, 40000]      *   |
     [9]     chr1       [40001, 45000]      *   |
     ...      ...                   ...    ... ...
[616079]     chrY [57745001, 57750000]      *   |
[616080]     chrY [57750001, 57755000]      *   |
[616081]     chrY [57755001, 57760000]      *   |
[616082]     chrY [57760001, 57765000]      *   |
[616083]     chrY [57765001, 57770000]      *   |
[616084]     chrY [57770001, 57775000]      *   |
[616085]     chrM [       1,     5000]      *   |
[616086]     chrM [    5001,    10000]      *   |
[616087]     chrM [   10001,    15000]      *   |

seqlengths
  chr1  chr2  chr3  chr4  chr5  chr6 ... chr20 chr21 chr22  chrX  chrY  chrM
    NA    NA    NA    NA    NA    NA ...    NA    NA    NA    NA    NA    NA
```

This example makes a `GRanges` object of 5 kb windows along all 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), 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
```

This example counts reads that fall within 2000 bases upstream and 500 bases downstream of the first six gene TSSs in the gene annotation table.

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 fragment 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, it is meant 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(5e+07,
    1e+08))
mappabilityCalc(locations, window = 500, organism = Hsapiens36bp)
```

```
[1] 0.000 0.998
```

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

The researcher might have a set of locations that they want to know the CpG density of.

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

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

This example calculates the CpG density of a window 100 bases either side of the TSS for the first six 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.

The package is still in active development, and near-term goals include more streamlining of the function signatures, and more analysis pipelines for sequencing data, including the use of input samples to remove genomic background from epigenomic signals.

8 Environment

This vignette was created in:

```
sessionInfo()
```

```
R version 2.13.0 (2011-04-13)
Platform: x86_64-unknown-linux-gnu (64-bit)

locale:
 [1] LC_CTYPE=en_AU.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_AU.UTF-8        LC_COLLATE=en_AU.UTF-8
 [5] LC_MONETARY=C              LC_MESSAGES=en_AU.UTF-8
 [7] LC_PAPER=en_AU.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_AU.UTF-8 LC_IDENTIFICATION=C

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

other attached packages:
 [1] BSgenome.Hsapiens36bp.UCSC.hg18mappability_1.0
 [2] gplots_2.8.0
 [3] caTools_1.11
 [4] bitops_1.0-4.1
 [5] gdata_2.8.1
 [6] gtools_2.6.2
 [7] zoo_1.6-4
 [8] edgeR_2.1.17
 [9] BSgenome.Hsapiens.UCSC.hg18_1.3.16
[10] BSgenome_1.19.6
[11] Biostrings_2.19.18
[12] GenomicRanges_1.3.38
[13] IRanges_1.9.31
[14] Repitools_1.91

loaded via a namespace (and not attached):
[1] Biobase_2.11.10 lattice_0.19-17 limma_3.7.27    tools_2.13.0
```