

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

Mark Robinson Aaron Statham Dario Strbenac

Last compiled on: April 18, 2011

1 Introduction

Repitools is a package that allows statistics of differential epigenetic marking 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. The utility of **Repitools** comes from that 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 checking of the sequencing data, followed by analysis and visualisation will be demonstrated. A more detailed description of the package can be found in the associated Bioinformatics Applications Note ¹

To start with, load the **Repitools** package.

```
library(Repitools)
```

2 Data

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")**.

```
load("samplesList.RData")
names(samples.list)
```

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

Also, an annotation of genes will be used. The annotation is based on one provided from Affymetrix with their expression arrays ².

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

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

```
gene.anno <- read.csv("geneAnno.csv", stringsAsFactors = FALSE)
head(gene.anno)
```

	name	chr	strand	start	end	symbol	selected_identifier
1	7896759	chr1	+	781253	783614	LOC643837	AK096570
2	7896761	chr1	+	850983	869824	SAMD11	NM_152486
3	7896779	chr1	+	885829	890958	KLHL17	NM_198317
4	7896798	chr1	+	891739	900345	PLEKHN1	NM_032129
5	7896817	chr1	+	938709	939782	ISG15	NM_005101
6	7896822	chr1	+	945365	981355	AGRN	NM_198576

Lastly, there is matrix of gene expression difference data, 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](#). The expression differences matrix will be used when illustrating some of the visualisation functionality later in the vignette.

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

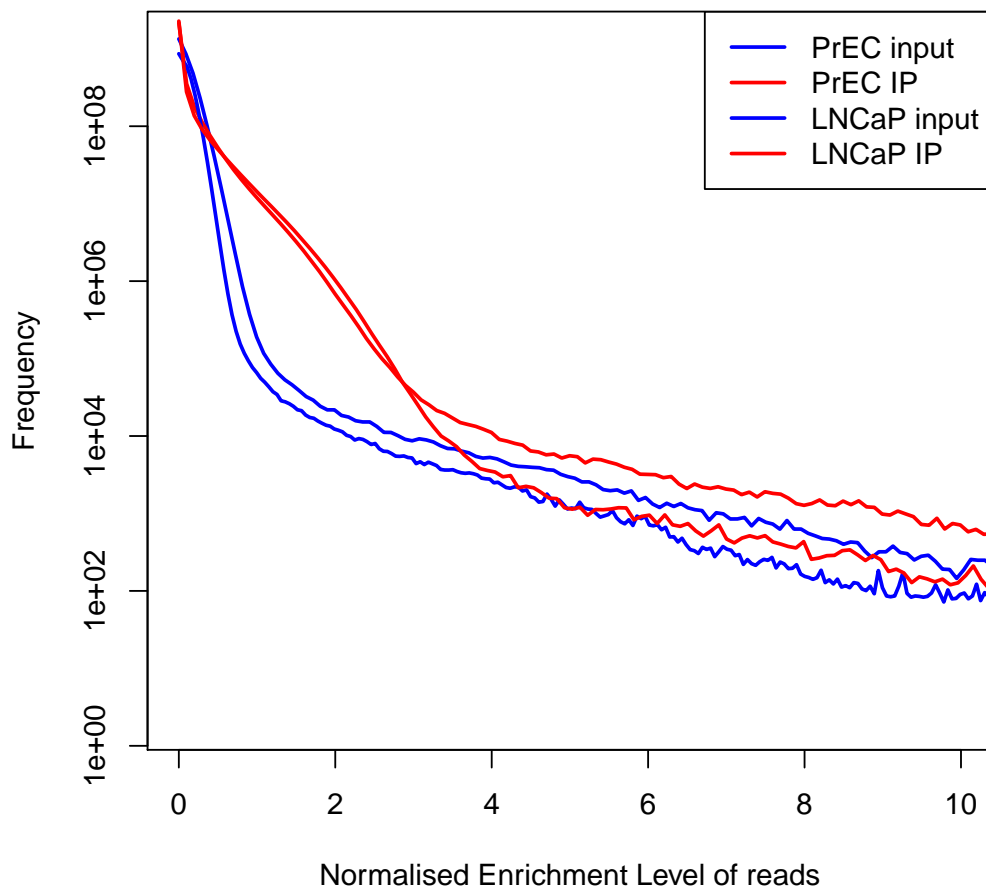
	t-stat
7896759	4.1130688
7896761	3.0691214
7896779	0.9724271
7896798	-0.5090460
7896817	2.1949896
7896822	-6.4049774

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.

```
enrichmentPlot(samples.list, 300, cols = c("blue", "red", "blue",
      "red"), xlim = c(0, 10), lwd = 2)
```

Enrichment Plot

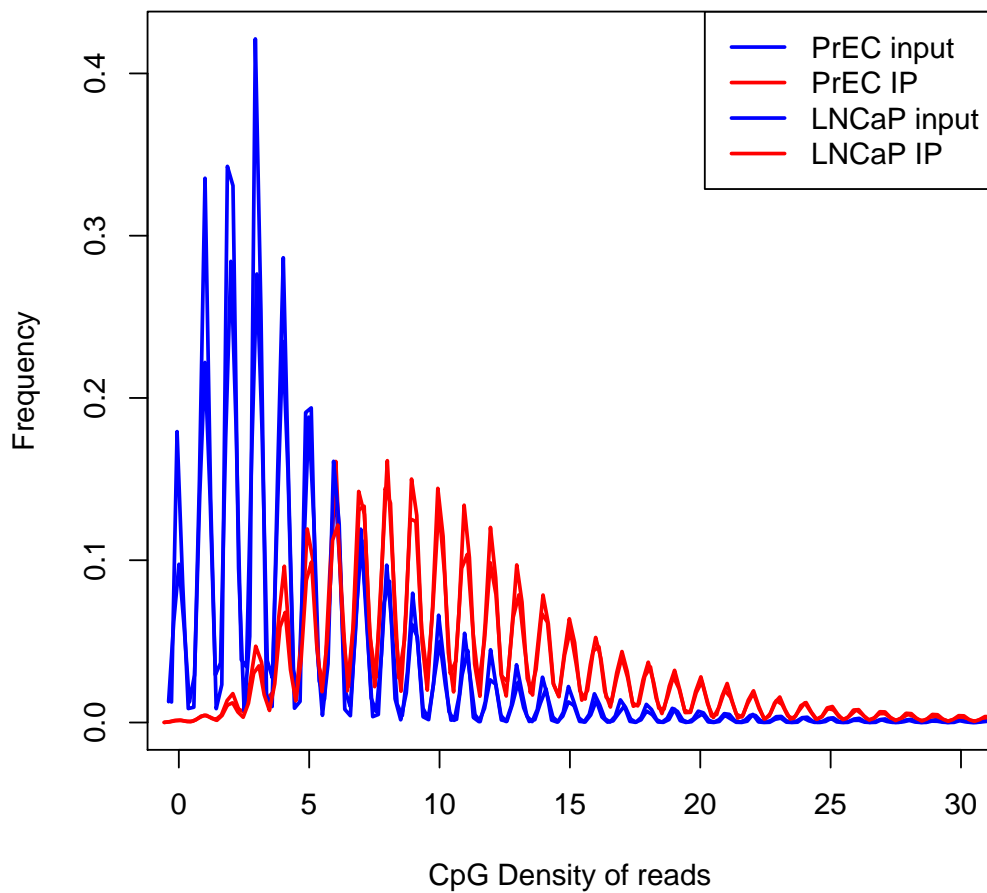


The above code uses the `Hsapiens` object to get the maximum base of chromosomes. The normalisation of the coverage used is to scale every coverage value by $10 \text{ million} / \text{number_of_reads_in_sample}$. 300 is passed in as the `seq.len` parameter, because that is approximately the real length 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("blue", "red", "blue", "red"), xlim = c(0,
    30), lwd = 2)
```

CpG Density Plot



This time the `Hsapiens` annotation is required so that the 300 base DNA sequence (tags are only 36 bp long) may be fetched. 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.

For more general sequencing quality checking, the FastQC ³ Java application has been gaining in popularity due to its speed and variety of results. A container class, `FastQC`, accessors, and the method `readFastQC` for reading in the raw Java program text file output and creating a `FastQC R` object, have been made to provide a framework for the fast development of quality control report generating pipelines. Higher level container classes with accessor methods are `SequenceQC`, which groups `FastQC` objects for the aligned and unaligned reads of a single sample together, and `SequenceQCSet` which is a collection of `SequenceQC` objects, perhaps of multiple sequencing samples within the same experimental run.

An example of turning a `SequenceQCSet` of the DNA methylation read data into a quality report PDF is demonstrated next.


³<http://www.bioinformatics.bbsrc.ac.uk/projects/fastqc/>

```
load("QCset.RData")
summary(QCset)
```

	Length	Class	Mode
PrEC input	1	SequenceQC	S4
PrEC IP	1	SequenceQC	S4
LNCaP input	1	SequenceQC	S4
LNCaP IP	1	SequenceQC	S4

```
pdf("QCreport.pdf", height = 8, width = 12)
genQC(QCset, "DNA Methylation Experiment")
dev.off()
```

pdf
2

Click here to see the report. 

When looking at the base distributions of the inputs, for either all of the reads, or the aligned subset, it can be seen that the frequency of A or T is always more than the G or C frequency. This mirrors the background distribution of the human genome. In the IP samples, the G or C frequency has risen, and the A or T has dropped, as would be expected when enriching for GC rich DNA methylated sequences.

This mismatches by cycles plots show that the IP samples tend to have a bias of G being called as T, and this is independent of the sequencing cycle. PrEC input has a tendency to miscall T as G more often than any other error, no matter the cycle.

4 Analyses and Visualisations

The `blocksStats` function is a convenient way to do a statistical test of differential enrichment between two groups or treatments, for counts in windows surrounding some genomic landmarks, like TSSs. The function leverages the package `edgeR`'s modelling of counts as negative binomial distributed and its adaptation of Fisher's exact test to overdispersed data.

```
design.matrix <- matrix(c(0, -1, 0, 1), dimnames = list(names(samples.list),
  "Cancer - Normal Methylation"))
design.matrix
```

	Cancer - Normal Methylation
PrEC input	0
PrEC IP	-1
LNCaP input	0
LNCaP IP	1

```
stats <- blocksStats(samples.list, gene.anno, 2000, 0, 300, design.matrix)
```

Comparison of groups: 1 - -1

```
stats <- stats[order(stats$`adj.p.vals_Cancer - Normal Methylation`),
]
head(stats)
```

	chr	start	end	width	strand	name	symbol
8019804	chr18	99064	112217	13154	+	8019804	ROCK1
8015798	chr17	38802738	38821439	18702	-	8015798	---
7904879	chr1	145017918	145018085	168	+	7904879	---
7908529	chr1	196148257	196165896	17640	+	7908529	LHX9
8115391	chr5	153834725	153838017	3293	-	8115391	HAND1
7976073	chr14	85066240	85164023	97784	+	7976073	FLRT2

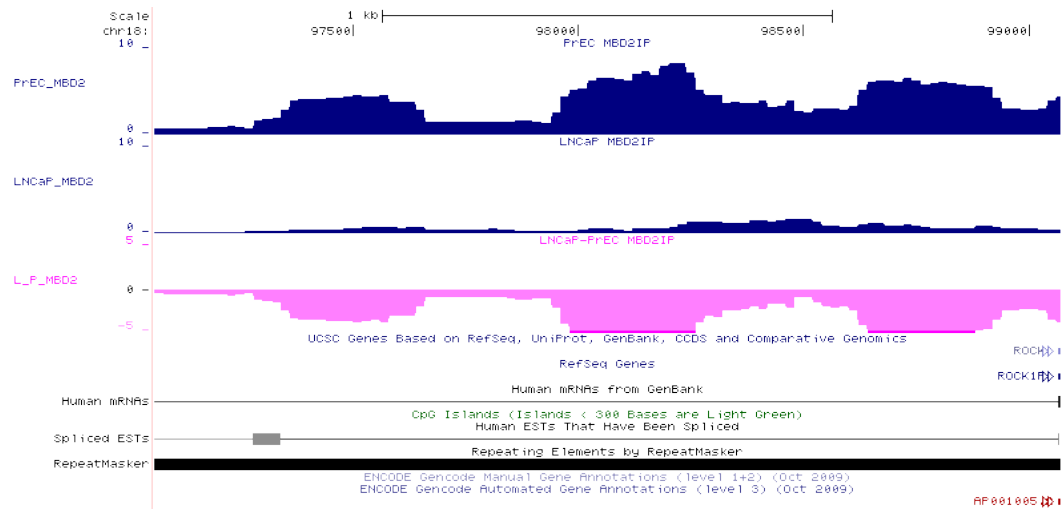
	selected_identifier	PrEC	input	PrEC	IP	LNCaP	input	LNCaP	IP
8019804	BC041849		600		397		686		58
8015798	BC035366		87		56		64		314
7904879	ENST00000364272		21		13		28		153
7908529	NM_001014434		16		3		13		112
8115391	NM_004821		8		4		28		95
7976073	NM_013231		17		0		20		69

	PrEC	IP_pseudo	LNCaP	IP_pseudo	logConc_Cancer - Normal Methylation
8019804	3.995887e+02		57.62379		-16.01856
8015798	5.636562e+01		311.96569		-16.21306
7904879	1.308530e+01		152.00848		-17.78513
7908529	3.020114e+00		111.27404		-19.06788
8115391	4.026631e+00		94.38415		-18.97912
7976073	1.435022e-11		68.55255		-33.59047

	logFC_Cancer - Normal Methylation	p.value_Cancer - Normal Methylation
8019804	-2.793764	9.642793e-36
8015798	2.468516	3.662695e-24
7904879	3.538199	3.672510e-18
7908529	5.203643	7.401219e-18
8115391	4.551106	2.875722e-14
7976073	32.851160	1.911561e-13

	adj.p.vals_Cancer - Normal Methylation
8019804	2.407420e-31
8015798	4.572142e-20
7904879	3.056262e-14
7908529	4.619471e-14
8115391	1.435905e-10
7976073	7.954003e-10

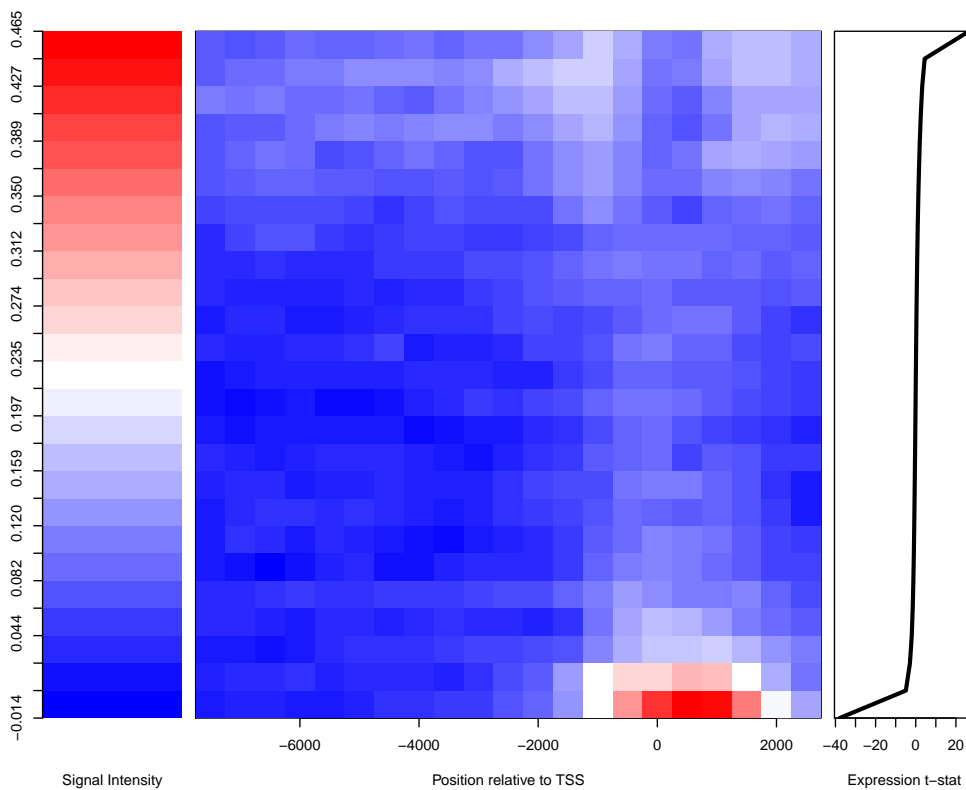
The example calculates statistics on TSS regions which start 2000 bases upstream of the TSS and finish at the TSS, after the reads have been extended to being 300 bases long. A coverage plot from UCSC browser illustrates the best found region. The coverage values are scaled to be as if there were 10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



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, by = 500,
  bw = 500, seq.len = 300, ordering = expr, ordLabel = "Expression t-stat",
  plotType = "heatmap", nbins = 25)
```

Signal:Cancer – Normal Methylation Order:t-stat

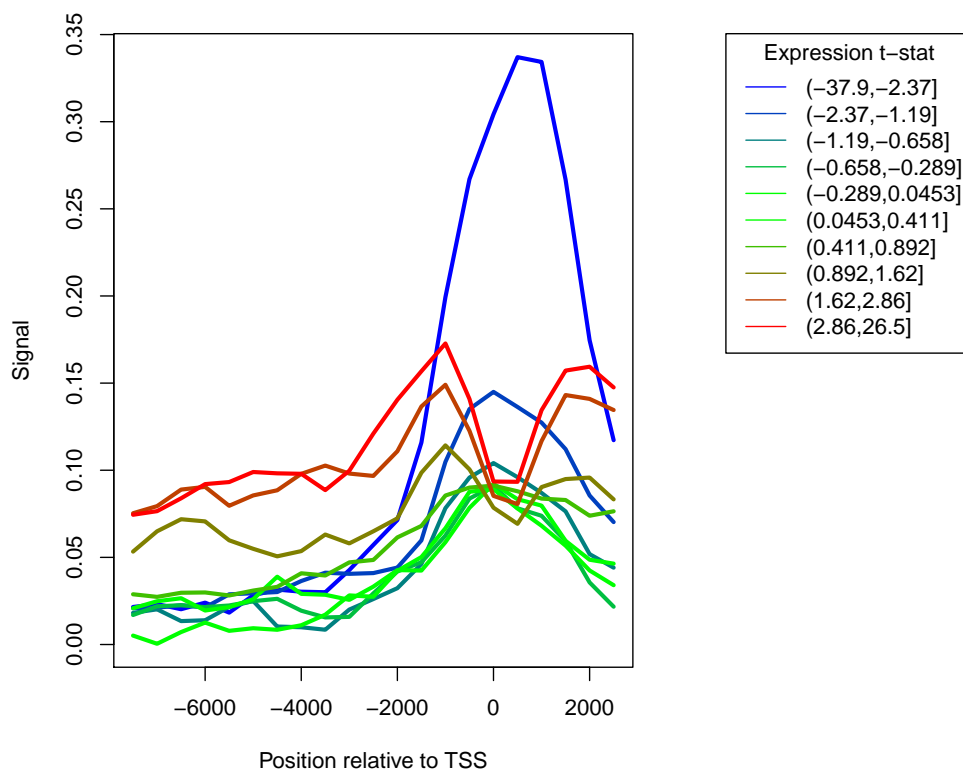


This example made counts in 500 base non - overlapping windows between -7500 bases upstream and 2500 bases downstream (the default range) for each gene, then split them into categories based on the expression difference value, and averaged over all counts for each particular window and expression category. 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, by = 500,
  bw = 500, seq.len = 300, ordering = expr, ordLabel = "Expression t-stat",
  plotType = "line", nbins = 10)
```

Signal:Cancer – Normal Methylation Order:t-stat

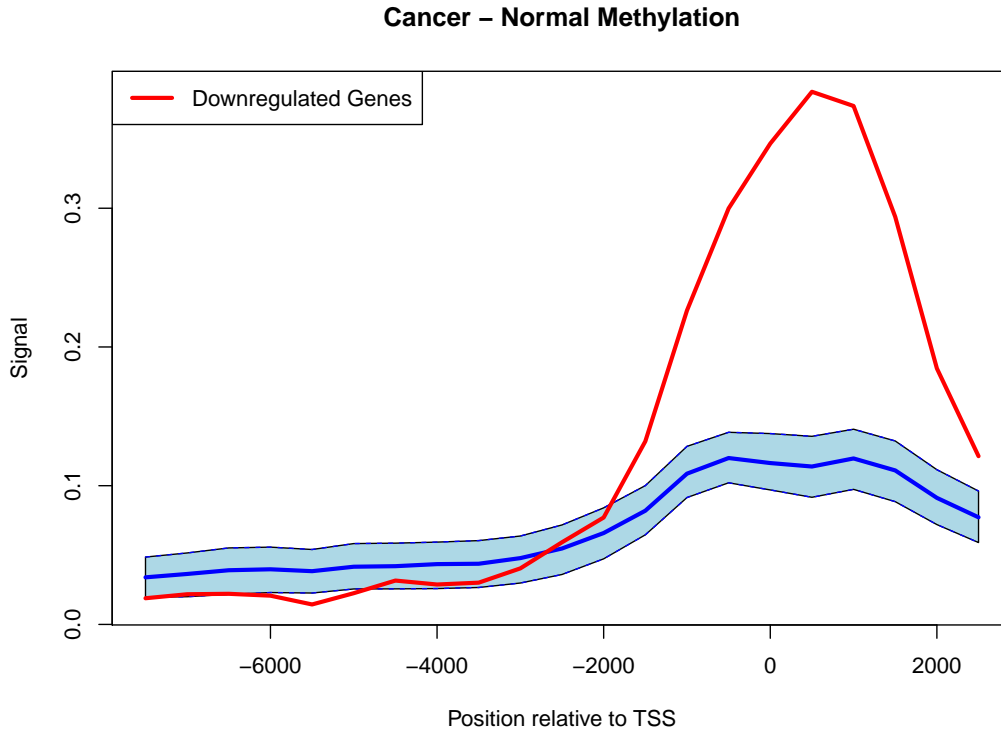


Some genes may be of interest to the researcher for some reason. This subset of genes may be known to be strongly marked with another epigenetic mark, or change in expression in the same direction strongly, or many other reasons. No matter what the reason for selecting the subset is, the profile of intensities 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, gene.anno, 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 expression-losing gene set has a significant gain of methylation 2000 bases either side of the TSSs, in comparison to random sets of other genes.

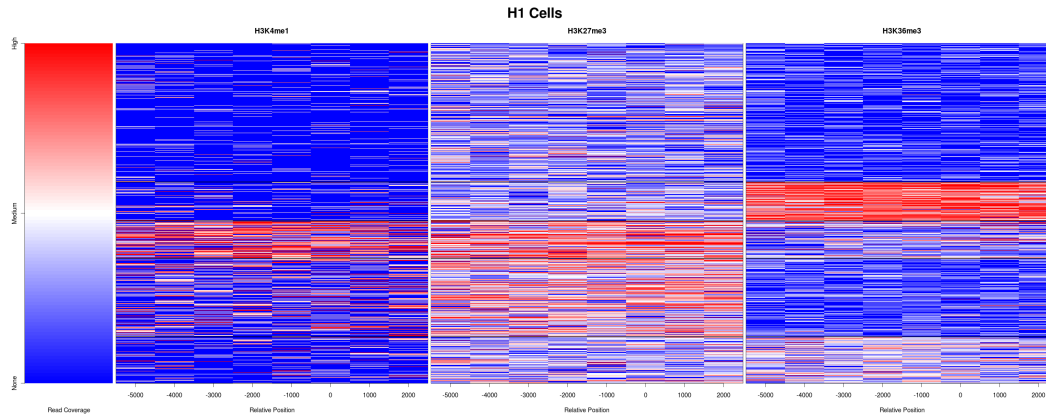
`clusterPlots` is another way to look at read depth at regular positions around a feature. The current usage of this function is for when there are multiple epigenetic marks for the same sample. The first step is to use `featureScores` to get the coverage tables, which essentially gives a list of coverage 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](#).

```

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

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

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 statistics 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, 8, 5, 2, 3, seq(2,
10, 2), trend = "up")
cluster.1 <- which(stats.clustered$cluster == 1)
stats.clustered[cluster.1, ]
```

	name	chr	strand	start	end	symbol	selected_idenfier
8038643	8038643	chr19	-	56020356	56026591	KLK15	NM_017509
8030753	8030753	chr19	+	56049982	56055832	KLK3	NM_001030047
8030768	8030768	chr19	+	56068500	56075635	KLK2	NM_001002231
8038653	8038653	chr19	-	56077163	56091466	KLKP1	NR_002948
8038655	8038655	chr19	-	56101419	56105806	KLK4	NM_004917
8038668	8038668	chr19	-	56122145	56122637	LOC390956	ENST00000324656
	t-stat	pos	cluster				
8038643	4.8240911	56026591	1				
8030753	15.0273556	56049982	1				

8030768	16.6680177	56068500	1
8038653	9.0615372	56091466	1
8038655	14.2031281	56105806	1
8038668	0.6970488	56122637	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, 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 negative t-statistic, if a consecutive run of at least 3 genes with t-statistic being positive could be made, then this forms a cluster.

5 Utility Functions

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

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

```
genomeBlocks(Hsapiens, 1:25, 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.

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.

`annotationCounts` is useful to understand the counts of reads surrounding some landmarks, like TSSs. `annotationBlocksCounts` is the analogous function for counting in user-specified regions of the genome.

```
annotationCounts(samples.list, head(gene.anno), 2000, 500, 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 genes in the gene annotation table.

It would be good 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, and it is common practice in the analysis of sequencing data to discard reads that map to multiple locations in the genome. 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, 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.

6 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.

7 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] zoo_1.6-4
[3] gplots_2.8.0
[4] caTools_1.11
[5] bitops_1.0-4.1
[6] gdata_2.8.1
[7] gtools_2.6.2
[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] Repitools_1.91
[14] IRanges_1.9.31
```

```
loaded via a namespace (and not attached):
```

```
[1] Biobase_2.11.10 lattice_0.19-17 limma_3.7.27
```