A collection of ChIP-seq analyses: exploration, relating signal to gene expression, region finding and differential analyses, relating regions to annotation

Mark Robinson
mark.robinson@imls.uzh.ch

June 22, 2013

Contents

1 Introduction

Buongiorno! This vignette is a first attempt at collecting materials for a variety of chromatin immunoprecipitation sequencing (ChIP-seq) or affinity enrichment analyses that can be done in Bioconductor, largely making use of the *Repitools* package. The field of ChIP-seq analyses is large and only a small representation of analyses are given here. Many of these analyses here are exploratory in nature and we only scratch the surface of what can or should be done. Regardless, the analyses presented here serve as a stepping stone to further steps, using the vast infrastructure within Bioconductor for dealing with various data types, integration with other sources of information (e.g. annotation) and statistical and modeling tools.

2 Libraries

This segment loads all the libraries that are necessary for the running of this document, so that users know what packages need to be installed.

```
library("GenomicRanges")
library("Repitools")
library("BSgenome.Hsapiens.UCSC.hg18")
library("chipseq")
library("matrixStats")
```

Refer to http://bioconductor.org/install/ for further instructions on installing Bioconductor packages (Note: be sure to use the latest version of R and Bioconductor). See the Section "Versions" at the end of this document for a full list of the versions being used for this exercise.

See Section "Ideas for further exercises" for useful additional packages in the context of ChIP-seq (or related) data analysis.

3 Some datasets to explore

For this exercise, we first load some pre-processed ChIP-seq and MBD-seq (capture of methylated DNA using the methylated binding domain) data that are useful for illustrating the summaries and plots:

```
load("d.Rdata")
```

The first dataset contains several ChIP-seq experiments for prostate epithelial (PrEC) cells. Specifically, the dataset includes a mix of reads of different lengths, is reduced to a subset of the genome in the interest of allowing exploration on low-memory computers and includes the following experiments: H3K27me3, H3K36me3, H3K4me3, CTCF and an INPUT (genomic DNA) control.

```
load("methGR.Rdata")
```

The second dataset contains MBD-seq data on experiments for prostate epithelial (PrEC) and control cells. This dataset also includes a mix of different read lengths, is reduced in the interest of speed and low-memory requirements and includes: PrEC INPUT (genomic DNA), MBD capture of the PrEC methylome regions of the PrEC and of fully methylated DNA (in-vitro-treated with SssI).

NOTE: You may wish to know how similar (real and presumably larger) datasets can be loaded in the same manner as below. We assume that users would start with a set of already-mapped reads (the mapping itself is not discussed here), perhaps in BED or BAM format. These can be easily imported into R using `rtracklayer::import` or `Repitools::BAM2GRanges`/`Repitools::BAM2GRangesList` from BED and BAM, respectively. Note that the latter will only work for single-end data, while if BED files are the input, then paired-end reads should already be represented as single fragments (although not many ChIP-seq experiments are run in paired-end mode). Notice also that, in the interest of memory, some of the functions discussed below do not require all reads to be loaded into memory; some operations can take place by calling the function on a character vector of BAM files. Despite this, some operations, depending on the dataset, may need a computing environment with larger memory.

The following commands are simply here for the user to explore a few aspects of the loaded data, to get a feel for what is contained in the pre-processed object (e.g. number and length of reads). When applied to another dataset, some of these will be useful spot checks to make sure that data was read in correctly and so on.

```
class(d)
```

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

```
class(d[[1]])
```

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

```
names(d)
```

```
[1] "PrEC_H3K27me3" "PrEC_H3K36me3" "PrEC_H3K4me3"  "PrEC_INPUT"
[5] "PrEC_MBD2IP"   "PrECp9_CTCF"
```

```
d
```

```
GRangesList of length 6:
$PrEC_H3K27me3
GRanges with 1598834 ranges and 0 metadata columns:
            seqnames               ranges strand
               <Rle>            <IRanges>  <Rle>
        [1]    chr15 [18260125, 18260174]      -
        [2]    chr15 [18260186, 18260235]      -
        [3]    chr15 [18260352, 18260401]      +
        [4]    chr15 [18260503, 18260552]      +
        [5]    chr15 [18260923, 18260972]      -
        ...      ...                  ...    ...
  [1598830]    chr22 [49585662, 49585711]      -
  [1598831]    chr22 [49589455, 49589504]      -
  [1598832]    chr22 [49589702, 49589751]      +
  [1598833]    chr22 [49591031, 49591080]      -
  [1598834]    chr22 [49591370, 49591419]      +

...
```

```
<5 more elements>
---
seqlengths:
      chr1      chr2      chr3      chr4 ...      chrX      chrY      chrM
 247249719 242951149 199501827 191273063 ... 154913754  57772954     16571
```

```
table( seqnames(d[[1]])  )
```

```
  chr1   chr2   chr3   chr4   chr5   chr6   chr7   chr8   chr9  chr10  chr11
     0      0      0      0      0      0      0      0      0      0      0
 chr12  chr13  chr14  chr15  chr16  chr17  chr18  chr19  chr20  chr21  chr22
     0      0      0 248494 252457 257368 215151 187109 217419 102579 118257
  chrX   chrY   chrM
     0      0      0
```

```
table( width(d)[[1]] )
```

```
     50
1598834
```

```
elementLengths(d)
```

```
PrEC_H3K27me3 PrEC_H3K36me3  PrEC_H3K4me3    PrEC_INPUT   PrEC_MBD2IP
      1598834       3874149       2588869       2255895       2813372
  PrECp9_CTCF
      2900809
```

```
lapply( d, function(u) table( width(u) ))
```

```
$PrEC_H3K27me3

     50
1598834

$PrEC_H3K36me3

     49
3874149

$PrEC_H3K4me3

     49
2588869

$PrEC_INPUT

     36
2255895

$PrEC_MBD2IP

     36
2813372
```

```
$PrECp9_CTCF

     49
2900809
```

Exercise: Understand what each of these commands returns and apply the same to the the `methGR` (second dataset). How many reads are there for each experiment? What are the read lengths? What chromosomes are covered?

In addition, we will later need some annotation and gene expression data:

```
load("anno.Rdata")
load("expr.Rdata")
```

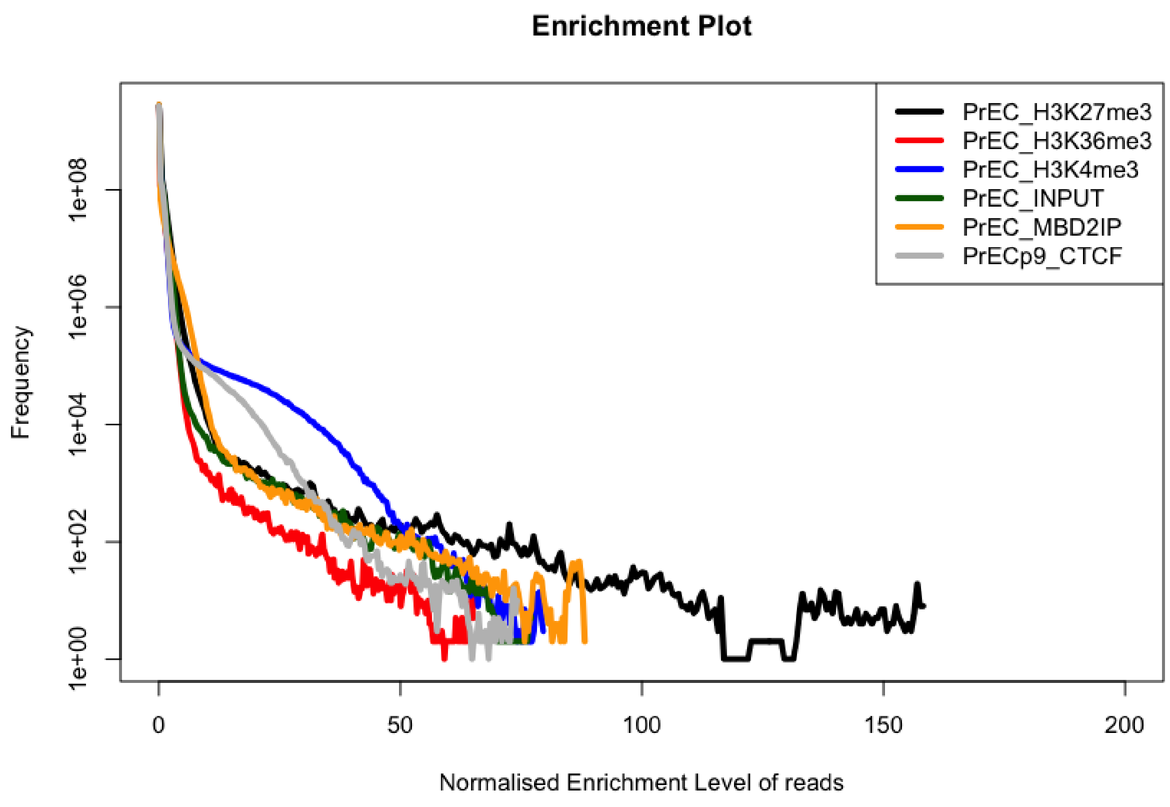
I encourage you to look and understand what this data consists of (e.g. using `head`, `dim`, `table`, etc.). Briefly, the annotation originates from an Affymetrix file and the `expr` is RMA-summarized-and-GC-adjusted expression signal from Affymetrix Gene 1.0 ST arrays.

4 Exploring the distribution of enrichment

One useful plot to explore the gross genome-wide relative enrichment of multiple datasets, which is useful for comparisons between epigenetic marks on the same cells as well as comparisons of the same epigenetic mark across cell types (or experimental conditions), is frequency versus coverage. In ChIP-seq experiments, many regions of the genome are not covered and the rate at which the coverage decays across experiments is different depending mainly on the diversity (e.g. how many regions are bound by a transcription factor, how many histone tails are in a particular state) of the captured library as well as on technical factors, such as antibody efficiency.

The `enrichmentPlot` function can be used on the `GRangesList` object:

```
cols <- c("black","red","blue","darkgreen","orange","grey")
e <- enrichmentPlot(d, seq.len=300, lwd=4, xlim=c(0,200), col=cols)
```



Under the hood, you can get an idea of what is calculated from this:

```
lapply(e[1:2],head)
```

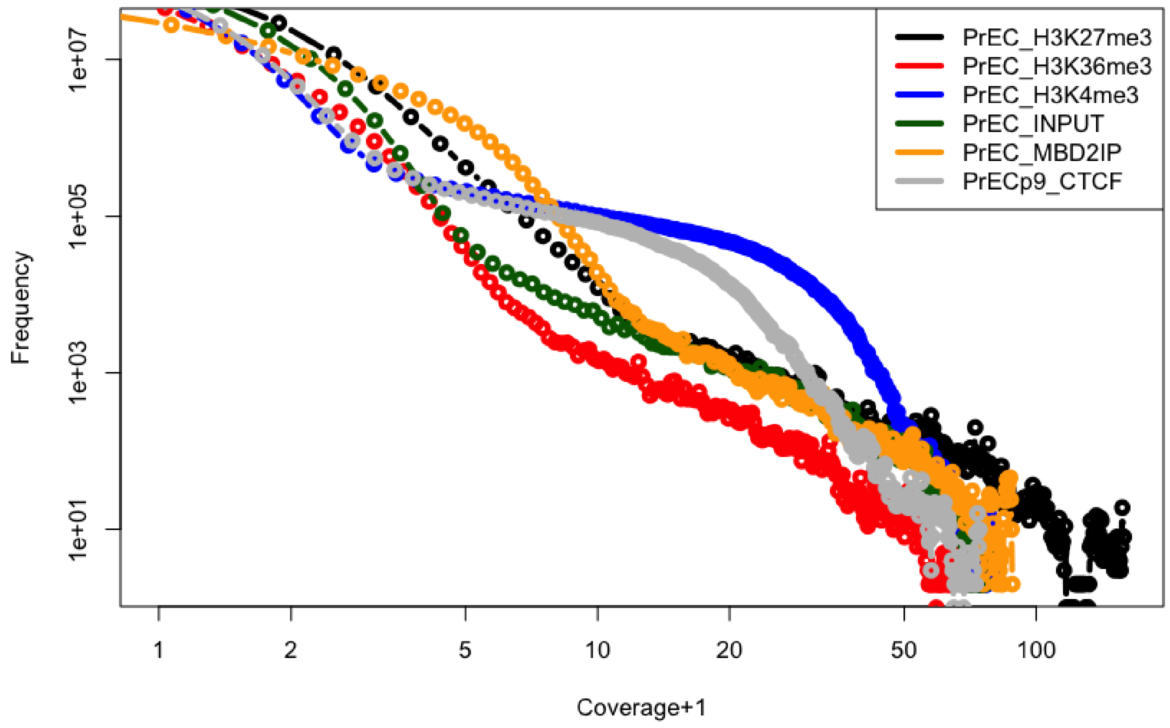
```
[[1]]
   coverage      bases
0 0.0000000 2809866899
1 0.6254558  150013808
2 1.2509116   71285157
3 1.8763674   29361284
4 2.5018232   11554262
5 3.1272790    4533644

[[2]]
   coverage      bases
0 0.0000000 2674188537
1 0.2581212  117019762
2 0.5162424  104721849
3 0.7743636   73916116
4 1.0324848   45563230
5 1.2906060   26403773
```

Note that the coverage here is already adjusted for the total depth.

A variation of this plot gives the coverage (X-axis) on a log scale, similar to plots that many people use for looking at power-law relationships, which certainly do not appear to happen in these instances.

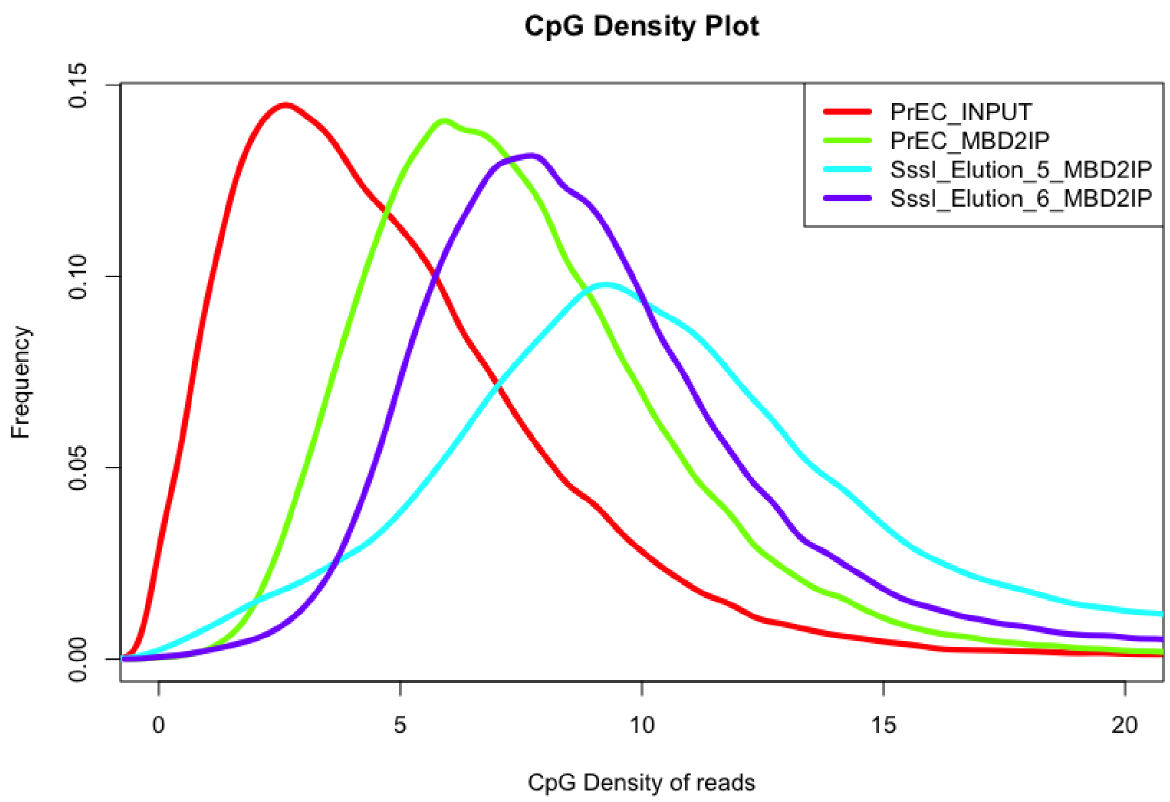
```
# set ranges to plot
xr <- range( unlist(lapply(e,".subset",1))+1 )
yr <- quantile(unlist(lapply(e,".subset",2)),p=c(.02,.98))
# Plot coverage frequency by ChIP experiment, manually
plot(xr*10,type="n",xlim=xr, ylim=yr,
     xlab="Coverage+1", ylab="Frequency", log="xy")
dummy <- sapply(1:length(e), function(u) {
  points( e[[u]]$coverage, e[[u]]$bases, col=cols[u], lwd=4, type="b")
})
legend("topright",names(d),col=cols,lwd=4)
```



Exercise: From these analyses, which epigenetic marks show high signal and which show more disperse signal. Is this expected? (Note: the answer to this requires some biological knowledge).

For methylation affinity capture data (e.g. MBD-seq), the above enrichment analyses are useful, but there are additional features that we can easily take advantage of. In particular, we expect to capture DNA in regions where methylation is expected (e.g. CpG islands). As yet another spot check of such data, we can look at the CpG density of the captured fragments. Specifically, we look at the CpG density of all reads (possibly extended to expected fragment length).

```
cpgDensityPlot(methGR, seq.len=300, organism=Hsapiens,
               lwd=4, w.function="linear", window=600, verbose=TRUE)
```



Exercise: Another control of interest is the CpG density of just the genome itself. Take a small sample of regions from the genome and compare it with the CpG density of the genomic DNA input. Is the genomic DNA INPUT representative of the genome? Hint: randomly select regions of the genome and add them to the `GRangesList` object and recalculate the CpG densities of the random locations, together with those from the input using `cpgDensityCalc` or `cpgDensityPlot`.

NOTE: the `cpgDensityCalc` and `cpgDensityPlot` function are not lightning fast and do not scale well to large datasets.

5 Looking at peaks/regions (and relationship between height and length) using a simple caller

First, let's define a simple function, which takes a `GRanges` object as input, and picks off a set of high coverage regions, according to some threshold. This isn't necessarily a great way to pick off enriched regions, but it serves the purpose of comparing "peak" shapes and sizes across epigenetic marks.

```
findRegionsSimple <- function(u,frag.len=300,lower=15) {
  # u is a 'GRanges' of reads
  u <- resize(u,frag.len)
  cv <- coverage(u)
  scv <- slice(cv, lower=lower)
  gr <- as(scv,"GRanges")
  mcols(gr)$view <- mcols(gr)$view + 1e6*as.numeric(seqnames(gr))
  gr <- unlist(reduce(split(gr,mcols(gr)$view)))
  mcols(gr)$mean <- unlist(viewMeans(scv))
  mcols(gr)$max <- unlist(viewMaxs(scv))
  gr
}
```

Using the Bioconductor *chipseq* package, we can get an estimate of the fragment size:

```
load("d.Rdata")
fr <- sapply(d, estimate.mean.fraglen)
fr
```

```
       PrEC_H3K27me3 PrEC_H3K36me3 PrEC_H3K4me3 PrEC_INPUT PrEC_MBD2IP
 chr15      241.3045      217.6382     199.4089   244.3569    172.7182
 chr16      236.5979      214.6161     184.9776   233.5614    147.9715
 chr17      236.6848      208.9306     168.2439   236.4944    155.8732
 chr18      242.8094      226.2935     218.6905   244.4996    161.7939
 chr19      235.8289      209.8650     141.3069   234.0066    149.8096
 chr20      237.3873      217.1464     195.8362   238.9443    158.3653
 chr21      238.2331      217.4095     208.2938   230.2794    143.1191
 chr22      239.7376      211.4404     173.1927   233.7997    150.3181
       PrECp9_CTCF
 chr15    220.7696
 chr16    217.8652
 chr17    212.6690
 chr18    226.7912
 chr19    207.1845
 chr20    218.3270
 chr21    220.6087
 chr22    212.6475
```

```
frm <- colMedians(fr)
frm
```

```
 [1] 237.8102 215.8813 190.4069 235.2505 153.0956 218.0961
```

We can apply our region calling algorithm to a `GRangesList` object, as follows:

```
regs <- mapply( function(u,v) {
  cat(".")
  findRegionsSimple(u,frag.len=round(v))
}, d, frm)
```

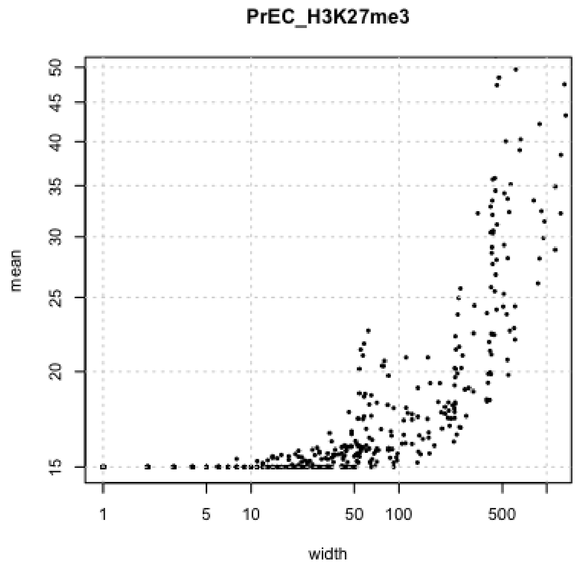
```
 ......
```

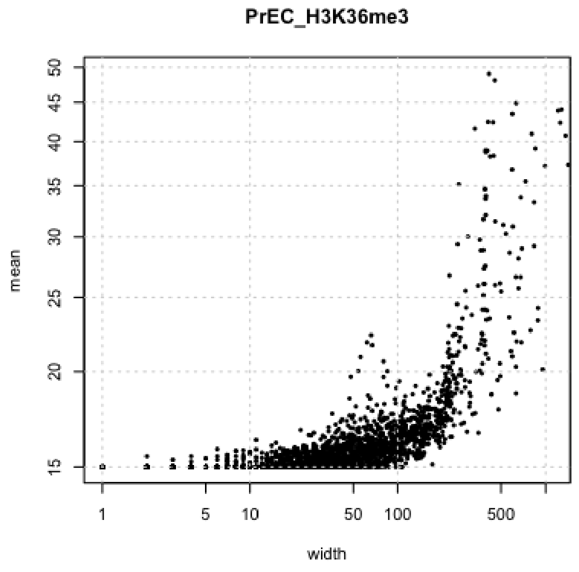
We can extract the called peak heights and widths, calculate some reasonable plotting limits:

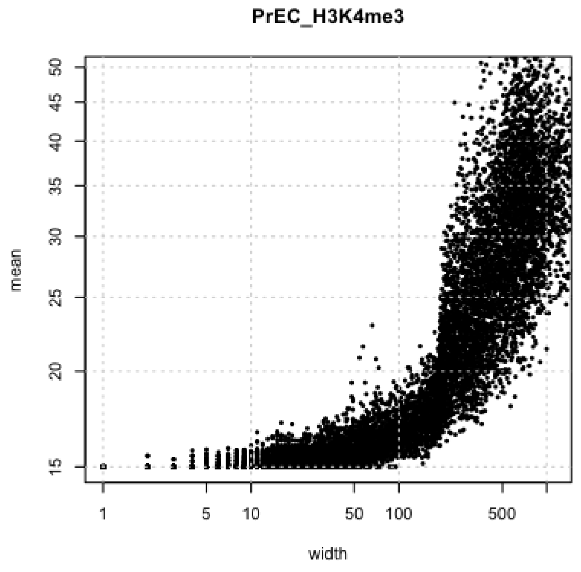
```
whs <- sapply(regs, function(u) {
  cbind(width=width(u),mean=mcols(u)$mean)
})
xlim <- quantile( unlist(lapply(whs,function(u) u[,1])), p=c(.01,.99) )
ylim <- quantile( unlist(lapply(whs,function(u) u[,2])), p=c(.01,.99) )
```

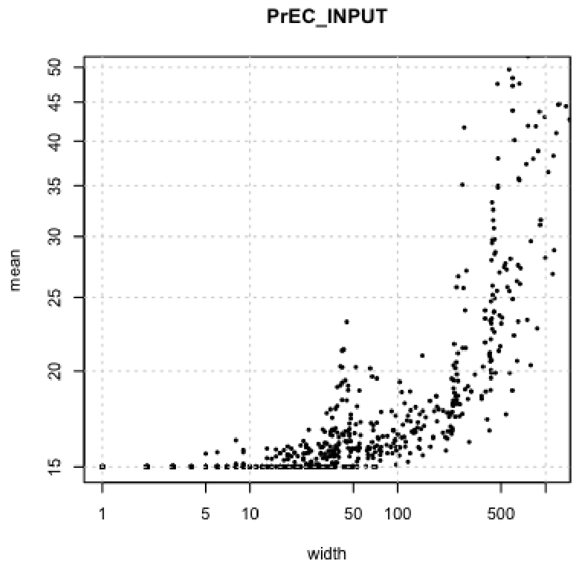
and then plot them:

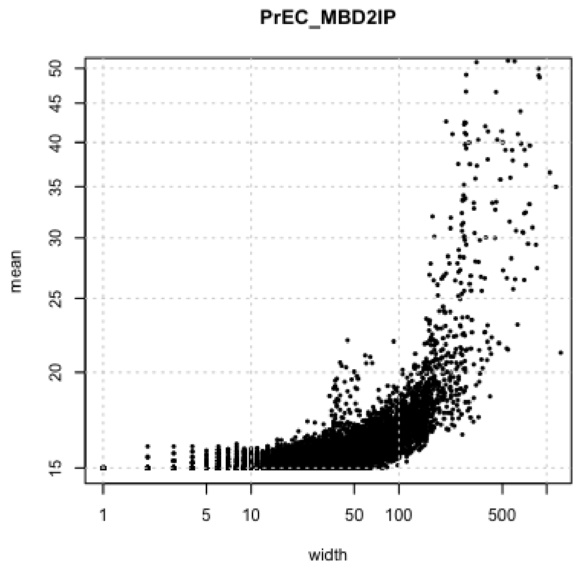
```
par(mfrow=c(3,2))
invisible(mapply( function(u,v) {
  plot(u,main=v,pch=19,cex=.4,log="xy", xlim=xlim, ylim=ylim); grid();
}, whs, names(whs)))
```

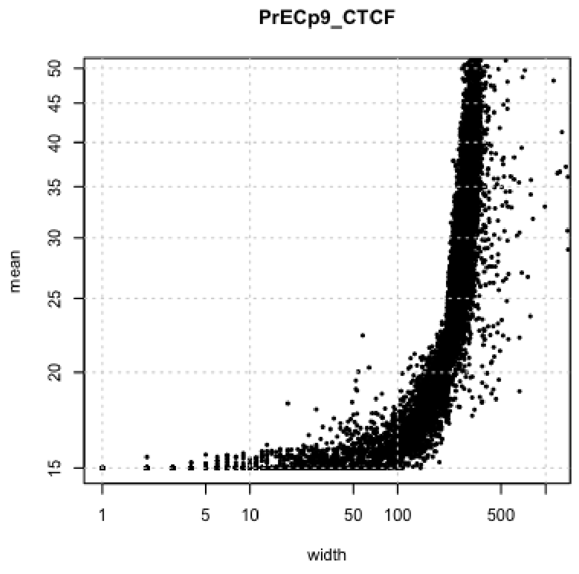

PrEC_H3K27me3
mean
width
15
20
25
30
35
40
45
50
1
5
10
50
100
500


PrEC_H3K36me3
mean
width
15
20
25
30
35
40
45
50
1
5
10
50
100
500


PrEC_H3K4me3
mean
width
15
20
25
30
35
40
45
50
1
5
10
50
100
500


PrEC_INPUT
mean
width
15
20
25
30
35
40
45
50
1
5
10
50
100
500


PrEC_MBD2IP
mean
width
15
20
25
30
35
40
45
50
1
5
10
50
100
500


PrECp9_CTCF
mean
width
15
20
25
30
35
40
45
50
1
5
10
50
100
500

Exercise: Study these plots and try to get a feel for whether "enriched" regions are disperse or punctate. Investigate with a more liberal or conservative peak calling.

6 Relating ChIP-seq signal to expression (or other) signal

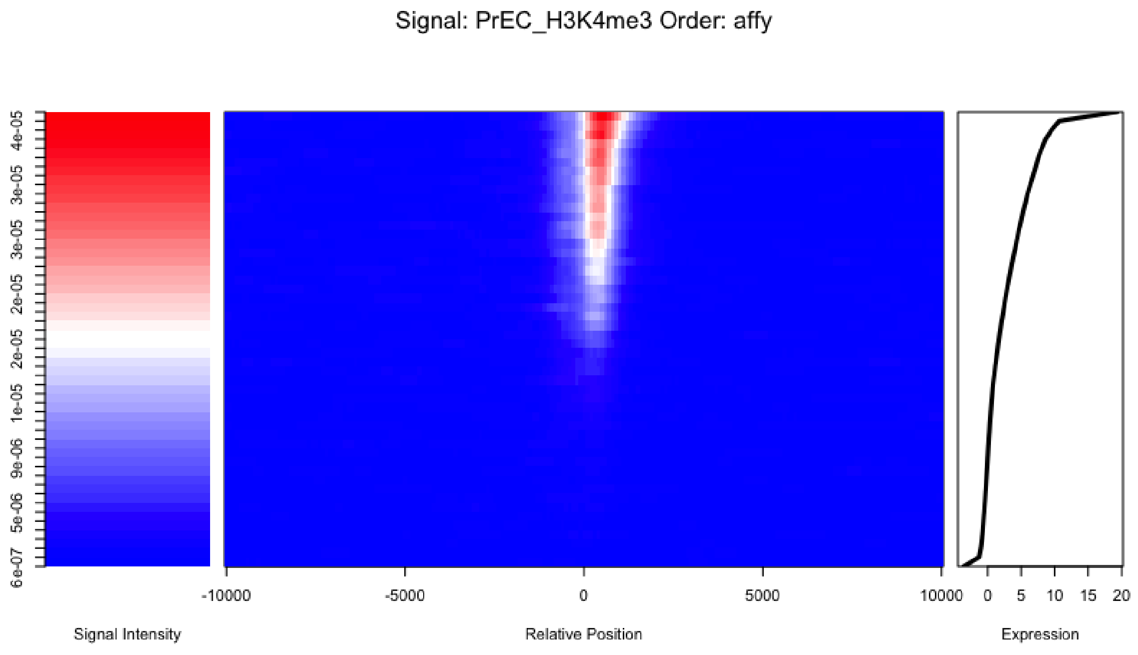
One thing that is useful as a spot check, but also as an informative analysis, is the association between the local ChIP-seq signal and the expression levels of the corresponding genes. Indeed, many things are already well known about these relationships, including a strong association between DNA methylation or H3K27me3 enrichment and a repressed state of expression and H3K4me3 enrichment and an active state. Therefore, exploring these relationships should be a standard way of looking at these datasets; the functions below facilitate this.

In *Repitools*, there is a general purpose function called `featureScores` that grabs ChIP-seq (or similar) signal, relative to given annotation anchor points (often transcription start sites). Users must specify what region up- and downstream of the anchor points, what resolution and what level of smoothing to apply, as follows:

```
covs <- featureScores(d, anno, up=10000, down=10000, freq=100, s.width=500)
```

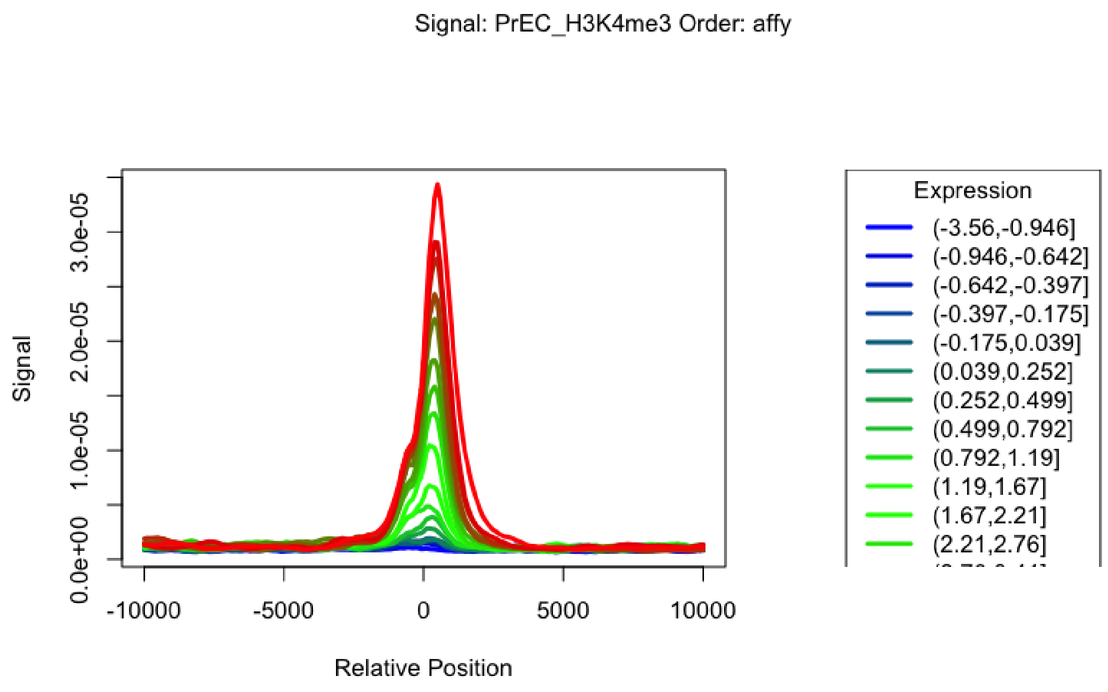
The output `ScoresList` is basically a matrix of signal with a row for each anchor and a column for each position up and down from the anchor point (one matrix for each element in the input `GRangesList`). Since this is in rectangular form, it can easily be related to a gene-level outcome variable, such as expression measurements from microarrays or sequencing. A useful summary of the relationship between ChIP-seq and expression can be conducted by putting the genes into ordered groups according to expression and averaging the corresponding ChIP-seq signal by group. This can be done and shown as a heatmap as follows:

```
binPlots(covs[3],ordering=data.frame(affy=expr),
         ord.label="Expression",n.bins=50,plot.type="heatmap")
```



This shows quite clearly that the groups of genes with the highest expression have a sharp H3K4me3 signal in the region very near to the TSS. The same plot can be visualized as a line plot:

```
binPlots(covs[3],ordering=data.frame(affy=expr),
         ord.label="Expression",n.bins=20,plot.type="line")
```



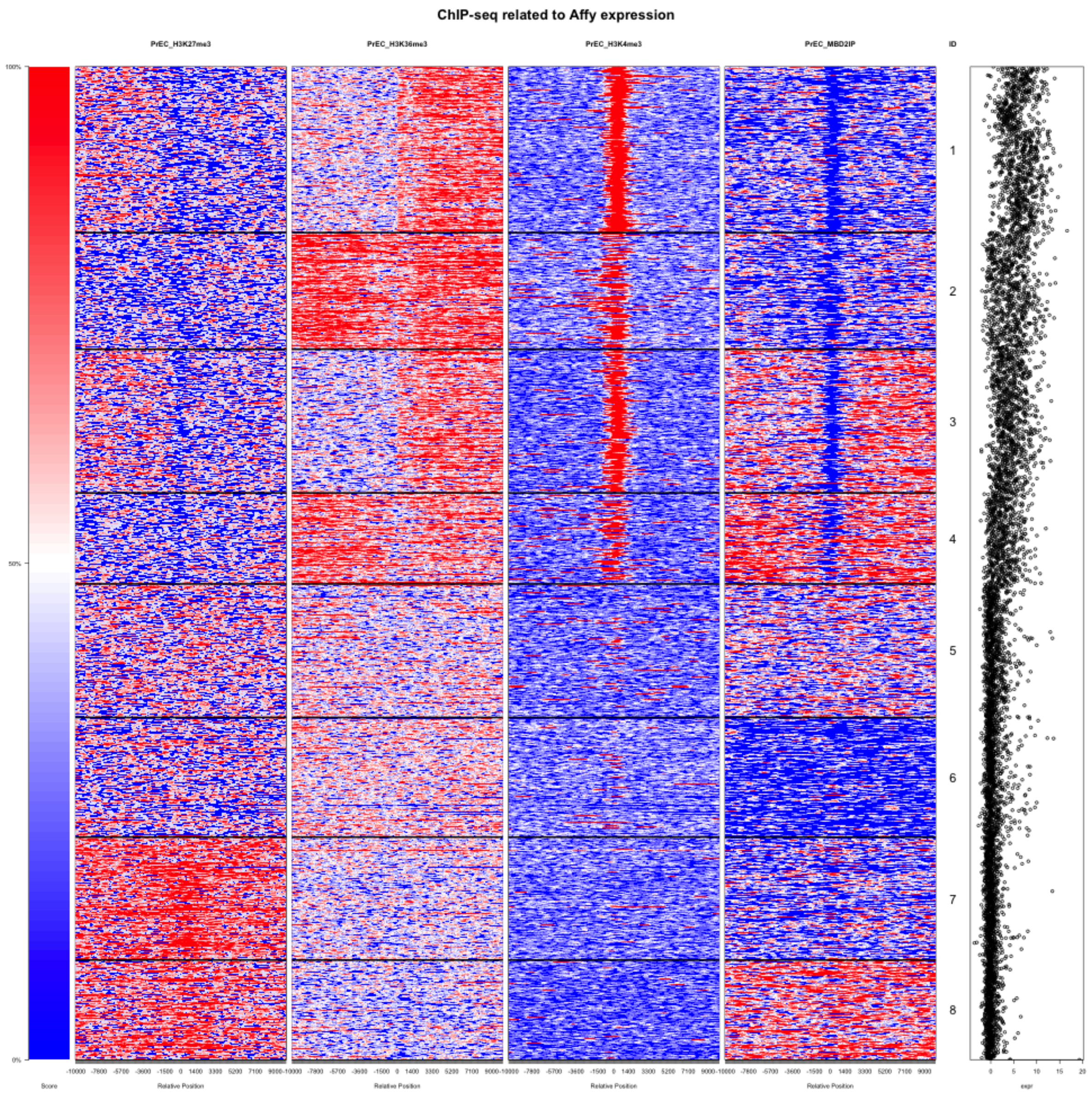
The same strategy can be applied to ChIP-seq (or similar) signal with any set of anchor points that have a corresponding numeric or categorical variable.

Exercise: Summarize the relationship between all the ChIP-seq signals and expression. You may need to modify the range of signal that is collected by `featureScores`. For H3K36me3, you may also need to play with the `dist` argument (of `featureScores`).

7 Clustering combinations of epigenetic signal

Another useful summary of ChIP-seq (or similar) data is the combination of signals that are present in the region surrounding the TSS. Scientists sometimes refer to the "histone code" to describe the (combinatorial) combination of epigenetic marks and how that relates to the regulation of the corresponding gene. A simple way to look at this is to take the signal from `featureScores` and cluster it. In *Repitools*, the function `clusterPlots` uses a standard k-means algorithm, since it is fast, as follows:

```
clusterPlots(covs[-c(4,6)], function(x) sqrt(x), expr=expr, plot.type="heatmap",
             t.name="ChIP-seq related to Affy expression", n.clusters=8)
```



Note that the expression data is not used here in the clustering, except to order the final clusters.

Exercise: Explore `clusterPlots` with `plot.type="line"`. Explore other numbers of clusters and other combinations of marks.

8 TSS read counts

Sometimes, it is useful to reduce ChIP-seq data to a number that can be compare to expression values or have statistical inference tools operate on. A simple and useful thing is counting the reads that fall in a region close to the TSS, for example:

```
ac <- annotationCounts(d, anno, up=500, down=500, seq.len=200)
head(ac)
```

```
        PrEC_H3K27me3 PrEC_H3K36me3 PrEC_H3K4me3 PrEC_INPUT PrEC_MBD2IP
7906303             1            13          193          4           0
7917645            11             2           13          3           1
7955117             1            15          178          4           0
7968270             4             8          144          5           0
7981748             1             3            3          2           0
7981773             6            15           12          7           1
        PrECp9_CTCF
7906303           6
7917645           4
7955117           5
7968270           9
7981748           2
7981773          15
```

See also the `annotationBlocksCounts` function, which is just a wrapper for the Bioconductor machinery to count reads that fall in blocks.

Exercise: Make some scatter plots (or boxplots in bins, etc.) of read counts of epigenetic marks in relation to themselves and in relation to the corresponding gene expression values.

9 Ideas for further exercises

If you are finished all of the above exercises, here is a list of some additional analyses that can be performed within Bioconductor:

1. Apply a proper peak calling algorithm, such as those within *PICS* or*BayesPeak*
2. Given a properly replicated dataset, apply differential analyses, such as those within *DiffBind* or *MMDiff*, or using `abcdDNA` within *Repitools*
3. Given a set of regions (either those enriched or differentially enriched), use *ChIPpeakAnno* to annotate the regions (e.g. find distance to neighbouring genes, etc.).

10 Versions

- R version 3.0.0 (2013-04-03), `x86_64-apple-darwin10.8.0`
- Locale: `de_CH.UTF-8/de_CH.UTF-8/de_CH.UTF-8/C/de_CH.UTF-8/de_CH.UTF-8`
- Base packages: base, datasets, graphics, grDevices, grid, methods, parallel, stats, utils
- Other packages: BiocGenerics 0.6.0, Biostrings 2.28.0, BSgenome 1.28.0, BSgenome.Hsapiens.UCSC.hg18 1.3.19, caTools 1.14, chipseq 1.10.1, cluster 1.14.4, gdata 2.12.0.2, GenomicRanges 1.12.4, gplots 2.11.0.1, gtools 2.7.1, IRanges 1.18.1, KernSmooth 2.23-10, lattice 0.20-15, latticeExtra 0.6-24, MASS 7.3-26, matrixStats 0.8.1, RColorBrewer 1.0-5, Repitools 1.6.0, Rsamtools 1.12.3, ShortRead 1.18.0
- Loaded via a namespace (and not attached): Biobase 2.20.0, bitops 1.0-5, edgeR 3.2.3, hwriter 1.3, limma 3.16.5, R.methodsS3 1.4.2, Rsolnp 1.14, snowfall 1.84-4, stats4 3.0.0, tools 3.0.0, truncnorm 1.0-6, zlibbioc 1.6.0