BayesPeak: Bayesian Analysis of ChIP-seq data

Jonathan Cairns and Christiana Spyrou

June 3, 2010

Introduction

BayesPeak is a Bioconductor package for the analysis of data sets from ChIP-seq experiments, particularly for identifying the genomic sites of protein–DNA interactions.

The algorithm models the positions and orientations of the sequenced fragments and determines the locations of enriched areas, corresponding to binding sites and histone modifications, by using a hidden Markov model and Bayesian statistical methodology.

The Bayesian approach to parameter and state estimation returns posterior probabilities as measure of certainty, and offers great scope for interpretation, as well as allowing for the use of these probabilities as weights in subsequent analyses (e.g. motif discovery).

The other important feature of the algorithm is the use of the negative binomial distribution to model the counts of sequenced reads. This allows for overdispersion and provides a better fit to the data than the Poisson distribution that has been widely used by other methods.

1 What is ChIP-seq?

Chromatin ImmunoPrecipitation (ChIP) is an experiment designed to study protein-DNA interactions, particularly to identify the genomic sites where proteins, such as transcription factors, bind to the DNA, or sites where histone modifications occur (D. Schmidt, M.D. Wilson, C. Spyrou, G.D. Brown, J. Hadfield, and D.T. Odom., 2009). The experiment produces samples that are enriched for the sites of interest compared to the rest of the genome. The use of this method combined with high-throughput sequencing of the samples is referred to as ChIP-seq.

Given our protein of interest, the ChIP-seq protocol usually consists of the following steps. (The exact protocol may vary between different experiments, but BayesPeak will still be able to perform the peak-calling step.)

- Cross-linking the proteins to the DNA - The protein is permanently bound to the DNA, usually with formaldehyde.
- Shearing - The cells are lysed, and the DNA is randomly cut into small fragments by sonication.
- Immunoprecipitation - An antibody specific to the protein of interest is used to isolate the protein and the attached DNA fragments. The resulting sample is enriched for those genomic regions. If we are instead interested in locating histones with a certain epigenetic modification, then we use an antibody specific to histones with that modification.
- Reverse crosslinking and purification - The bonds between the protein and DNA are broken. The DNA is subsequently purified.
- Sequencing - During library preparation the contents of the samples are size selected such that the fragments' length lies in the region of 200-300 bp. This step is required by the sequencing protocol.

Adaptors are attached to the fragments and amplification usually takes place. Subsequently, the sample undergoes "high-throughput sequencing" during which the sequences of the ends of the present fragments are identified. For ChIP-seq applications usually one end of each fragment is sequenced to produce "single-end" reads.

- Alignment - The DNA is aligned back to reference genome, taking the quality of the reads into account. Usually only the reads that map to unique locations in the genome are included in the downstream analysis.
- Analysis - The sites of interest correspond to the genomic regions where there is high abundance of reads compared to the background or the control sample. BayesPeak performs this identification step, also referred to as "peak-calling".

Usually this process is repeated omitting the immunoprecipitation step to produce a sample with no preferential enrichment. This control sample has the same characteristics as ChIP-seq data and its inclusion is important to identify experimental biases.

There are many sources of error - for example, misalignment, impurities, or DNA which simply has a high affinity for being sequenced - which can result in noise across the genome, or even false peaks. The BayesPeak model is designed to separate the peaks from the noise, and to avoid calling false peaks.

2 Simple workflow

Load the package as follows:

```
> library(BayesPeak)
```

The example data set used below, consisting of the files "H3K4me3-chr16.bed" and "Input-chr16.bed", can be downloaded from
`http://www.compbio.group.cam.ac.uk/Resources/BayesPeak/csbayespeak.html`.

The following code is a very simple example of a BayesPeak workflow, where we analyse the region 92,000,000 - 95,000,000 bp on chromosome 16. It should take a couple of minutes on a relatively modern machine.

```
> raw.output <- bayespeak("H3K4me3-chr16.bed", "Input-chr16.bed",
+                         chr = "chr16", start = 9.2E7, end = 9.5E7, job.size = 6E6)
> output <- summarise.peaks(raw.output, method = "lowerbound")
```

- `bayespeak()` runs the actual BayesPeak algorithm on the data. The two input files are bed files located in the working directory, with H3K4me3 being the IP-treated data set and Input ID being the control data. In each case, we could have provided a data.frame or a RangedData object (from the IRanges package) instead of a file path.

  In a valid bed file, each row of the file to contains information for a read. In particular, the chromosome, start position, end position and DNA strand appear in the 1st, 2nd, 3rd and 6th columns respectively.

  The function applies the algorithm to 6 Mb partitions of the genome (or "jobs") by default, as explained in section 3.

- `raw.output` is a list - it contains not only the bins called, but also some useful QC information (such as the model fit - in particular, this can be used to spot unreliable jobs as described in section 8). This output needs to be summarised.

- `summarise.peaks()` is used to summarise the `raw.output` object. This consolidates the raw bin calls into peaks and combines data across jobs.

We can analyse all of the data present in the .bed file with the following command, although this will take somewhat longer.

```
> raw.output <- bayespeak("H3K4me3-chr16.bed", "Input-chr16.bed")
> output <- summarise.peaks(raw.output, method = "lowerbound")
```

A parallelisation strategy is available to reduce the running time of this process, which is given in section 4.1. We now go into this workflow in more depth.

3 The algorithm

BayesPeak fits a Markov model to the data (the aligned reads) via Markov Chain Monte Carlo (MCMC) techniques.

The genome is firstly divided up into "jobs", i.e. short regions, by default of size 6 Mb, on which the algorithm is run independently. This allows us to account for the variation in read abundance across each chromosome.

Within a job, we divide the region into small bins (by default, 100 bases each), and we consider the number of reads whose starts lie within each bin, for each strand.

A hidden Markov model is fitted to the bins, thereby classifying them as enriched or unenriched for sites of interest. However, since the parameters of the model are unknown (for example, the mean number of counts within enriched or unenriched bins), we estimate them by sampling from their posterior distributions using MCMC methods.

The output of the algorithm is the Posterior Probability (often abbreviated to PP) of each bin being enriched. The PP value is useful not only for calling the peaks, but could also be used in downstream analysis - for example, to weight observations when searching for a novel transcription factor motif. The PP value is not to be confused with the p value from hypothesis testing.

For a full description of the model, please refer to the (C. Spyrou, R. Stark, A.G. Lynch, and S. Tavaré., 2009).

4 The `bayespeak()` function

The `bayespeak()` function performs the algorithm described above on two sets of data - treatment, and control (optional), which should be supplied in bed format. Each of these can be specified as a file location, a data.frame containing the columns "chr", "start", "end", "strand" (specified as forward, "+", or reverse, "-"), or a RangedData object from the IRanges package in BioConductor. For example:

```
> raw.output <- bayespeak("H3K4me3-chr16.bed", "Input-chr16.bed")
```

As mentioned in section 3, we break down the chromosome into "jobs", which by default are of length 6 Mb, and run the algorithm on each job to account for the variability of conditions across the chromosome. Thus, each job has its own set of associated parameters.

Within each job, read abundance is modelled using non-overlapping bins. To avoid missing any peaks that straddle a boundary between two bins, we run the algorithm a second time. This second job is described as "offset" and shifts the boundaries of the bins by half a bin's length.

The output of this function is a list of three things:

- `raw.output$peaks`: Locations of "potentially enriched" bins, with their associated posterior probabilities. (A "potentially enriched" bin is defined as any bin with $PP > 0.01$.) Note that this is output is preliminary and does not correspond to the final result of the analysis.
- `raw.output$QC`: Information about the individual jobs.
- `raw.output$call`: A record of the arguments used when the function was called.

4.1 Parallelising BayesPeak - use of the multicore package

Due to its computational intensity, BayesPeak can be slow. However, the jobs can be run in parallel. This allows us to take advantage of multiple processors and dramatically reduce the time the algorithm takes.

This feature requires the multicore package to be installed. At time of writing, it can be downloaded from `http://www.rforge.net/multicore/`.

Having installed multicore, you can run the `bayespeak()` function in parallel by using the `use.multicore = TRUE` option. You can override the number of cores that multicore uses with the `mc.cores` argument.

```
> library(multicore)
> raw.output <- bayespeak("H3K4me3-chr16.bed", "Input-chr16.bed", use.multicore = TRUE, mc.cores = 4)
> output <- combine.peaks(raw.output, method = "lowerbound")
```

5 The `summarise.peaks()` function

The raw output of `bayespeak()` consists of the details of all of the individual bins in each individual job. The function `summarise.peaks()` combines the output of the individual jobs and joins adjacent bins into contiguous regions to give the final peak calls.

In more detail, `summarise.peaks()` does the following:

- Filtering of unenriched jobs: The model naturally tries to identify enriched states in the background even for jobs with consistently low read abundance. This can result in many unreliable calls and, to avoid this, all calls associated with such jobs can be removed. (This issue, "overfitting", is described in detail in section 8.)
- Filtering of unenriched bins: We remove all bins whose PP values are below a certain value (see section 7).
- Assembly of enriched bins: The bins across all remaining jobs are collected together. If two jobs call exactly the same bin, which could happen in regions where jobs overlap, then the one with the larger PP value is used.
- Conversion of bins to peaks: Where a number of bins form a contiguous region, we combine them into one large peak. The large peak now is assigned a PP value, based on the PP values of its component bins. Combining the PP values from multiple bins may be done in several ways. As a default, the "lowerbound" method is used, which calculates a lower bound for the overall PP by a dynamic programming technique. (See the help file for more information.)

6 Examining the results

The quality of ChIP-seq samples should be monitored to ensure that the experimental procedure was successful, in terms of the immunoprecipitation, the library preparation and the sequencing. Occasionally, very low read abundance in the data sets may be related to a failed step in the process and will lead to unreliable findings when the experimental noise and signal cannot be distinguished.

Moreover, care should be taken when even successful ChIP-seq samples are analysed and the results are interpreted. In the following sections we discuss the stringency with which the potential peaks are chosen and how to avoid false peaks being called in regions of "noisy background". Enrichment varies along the genome and is naturally low in long regions including the centromere and the ends of the chromosomes. If an entire job is contained in such a region, the model will still try to identify peaks and in the next sections we explain how to identify and filter out these false discoveries.

7 Choosing an appropriate *PP* threshold value

Selecting the threshold for *PP* values (i.e. the threshold argument in `summarise.peaks()`) is an important step in the analysis, and its choice will affect the peaks returned. First, we will observe how the *PP* values are distributed within jobs.

For this section, we load the example `raw.output` object included with the package:

```
> data(raw.output)
> raw.output$call

bayespeak(treatment = "~/bed/MCF/MCF7_ER.bed", control = "~/bed/MCF/MCF7_Input.bed",
    use.multicore = TRUE)
```

(Note that although `raw.output` was generated from running `bayespeak()` with the above arguments, `raw.output$peaks` has been manually reduced to only contain the data from chr16 to save space.)

>From this object, extract the *PP* values for each job with the following:

```
> PP <- split(raw.output$peaks$PP, raw.output$peaks$job)
```

Recall that any bin with $PP < 0.01$ was removed during the initial analysis. Therefore, to show the full range of the *PP* values, we assume that all of the missing bins have $PP = 0$ and reinstate them:

```
> bin.width <- raw.output$peaks$end[1] - raw.output$peaks$start[1]
> job.bins <- (raw.output$QC$end - raw.output$QC$start)/bin.width
> job.bins <- job.bins[as.integer(names(PP))]
> for(i in 1:length(PP))
+ {
+         PP[[i]] <- c(PP[[i]], rep(0, job.bins[i] - length(PP[[i]])))
+ }
```

We can view each job in turn as follows:

```
> par(mfrow = c(2,2), ask = TRUE)
> for(i in 1:length(PP)) hist(PP[[i]], breaks =150, main = names(PP)[i])
```

Examining the plots, they appear to have two distinct behaviours, as explained below.

7.0.1 High density of reads

For example, consider the *PP* profile of job 324:

```
> i <- 9
> hist(PP[[i]], breaks =150, main = names(PP)[i], ylim= c(0,50))
```

The plot is given in figure 1

For data with a high density of reads, peaks are clearer and thus the *PP* values of bins converge to 0 or 1, as observed in the figure above. So, the default *PP* threshold value of 0.5 will suffice to distinguish between enrichment and unenrichment. This is the expected outcome from jobs that include enriched peaks.

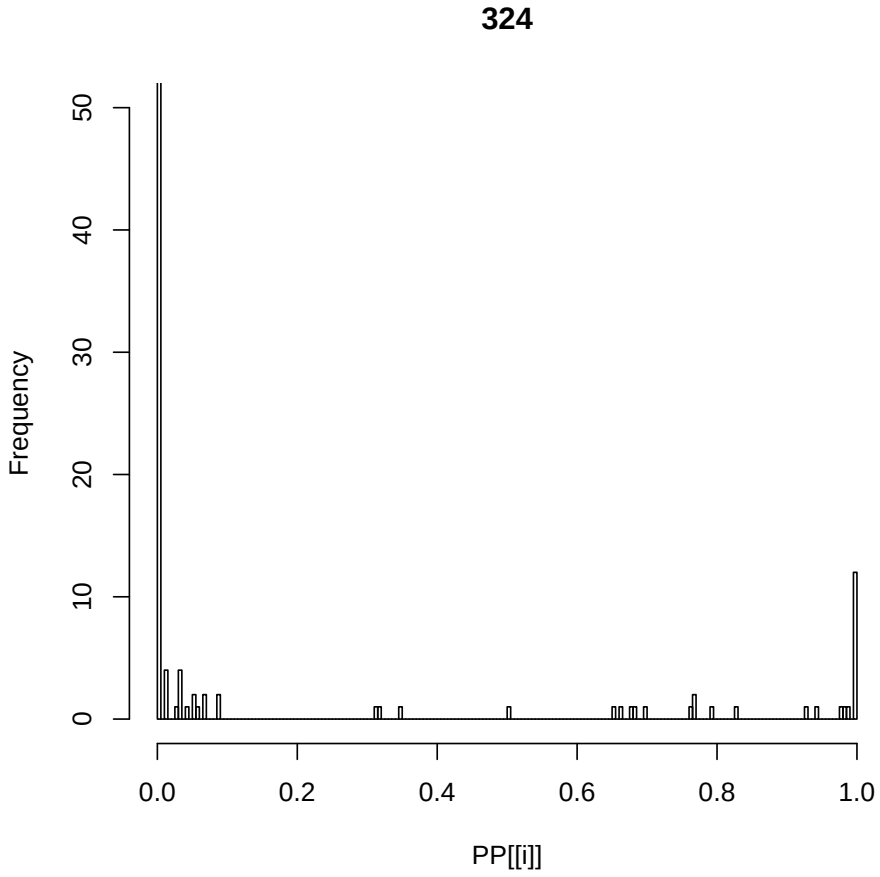

Figure 1: A histogram of the *PP* values found in job 324. Nearly all of the bins have a *PP* of 0 or 1.

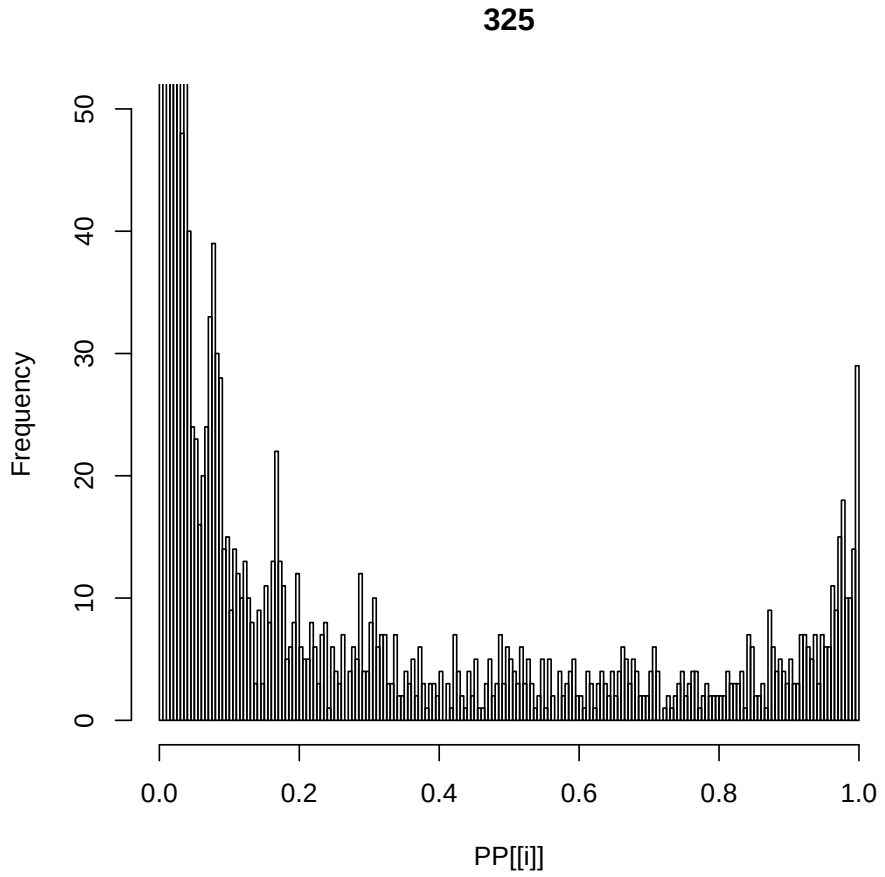

Figure 2: A histogram of the *PP* values found in job 325. The PP values are more evenly distributed over the interval (0,1) than they are in figure 1.

7.0.2 Low density of reads

On the other hand, as expected before, some jobs fall in areas of low enrichment and this affects their results. We demonstrate this by considering the *PP* values across job 325:

```
> i <- 10
> hist(PP[[i]], breaks =150, main = names(PP)[i], ylim= c(0,50))
```

The plot is given in figure 2.

When the coverage is sparse and therefore less information is available, the *PP* values tend to be more uniformly spread over the interval [0,1], as above. This means that the distinction between peaks and background is harder to make, which is usually a result of poor enrichment, if any. Such a uniform trend may be indicative of overfitting - we explore this effect in section 8.

In these jobs even the bins with highest *PP* cannot be classified as enriched since they are likely to correspond to random noise and, instead of choosing a high *PP* threshold, the whole job should be considered as unenriched.

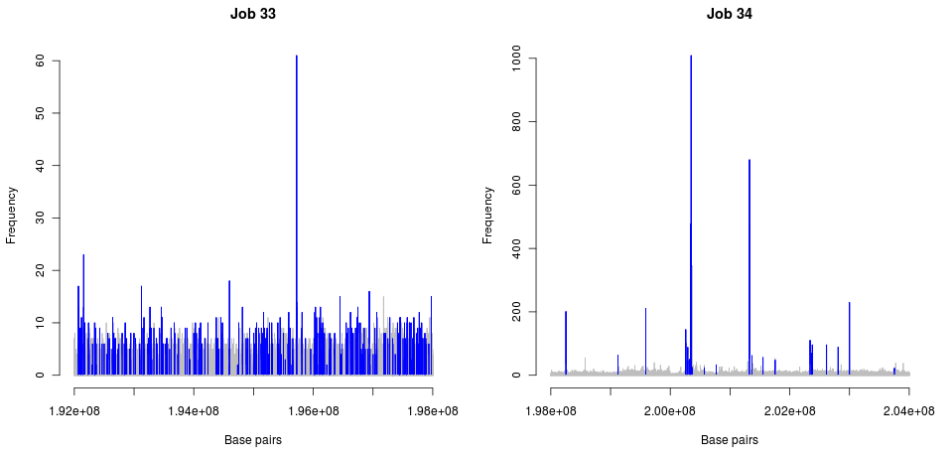

Figure 3: An example of overfitting. In each of these histograms, only reads on the positive strand are shown for clarity. A blue bar indicates that BayesPeak has called that bin as containing a peak at $PP > 0.5$. On the left, BayesPeak uses the enriched and unenriched states to explain the variance present in the background. On the right, the algorithm correctly identifies the large peaks present in the region of interest. (Notice the difference in scale between the two plots.)

8 Overfitting

BayesPeak can run into an overfitting problem when a job does not contain any peaks, or when peaks are weak compared to the background.

For this section, we again load the example `raw.output` object included with the package:

```
> data(raw.output)
```

The model assumes that there are both unenriched and enriched regions present. When the data contains no enriched regions, the model still tries to identify peaks in the data. Since some bins in the background will have higher counts than others, purely by chance, these will be marked as enriched. (See figure 3.)

This effect will be reflected in the parameters of the model, since the expected number of reads allocated at "enriched" regions will be much lower for the jobs where no peaks are present compared to the other jobs. This is one purpose of the QC component of the output - we can diagnose which peaks were called simply because they are in an unenriched job rather than because they are actual peaks.

Jobs suffering from this problem tend to exhibit three properties:

- Unusually large number of *calls*.
- Low mean number of counts in an enriched bin, defined as λ_1 in the model description in (C. Spyrou, R. Stark, A.G. Lynch, and S. Tavaré., 2009).
- PP values spread out over the interval [0,1] rather than mostly falling at 0 or 1, as explained in section 7. We quantify this using a score: of the bins with PP values > 0.01, the score is the proportion that have $PP > 0.5$. A low score is therefore indicative of overfitting.

We can observe when overfitting has occured in our data set by plotting these properties against each other. We generate the plots with this code:

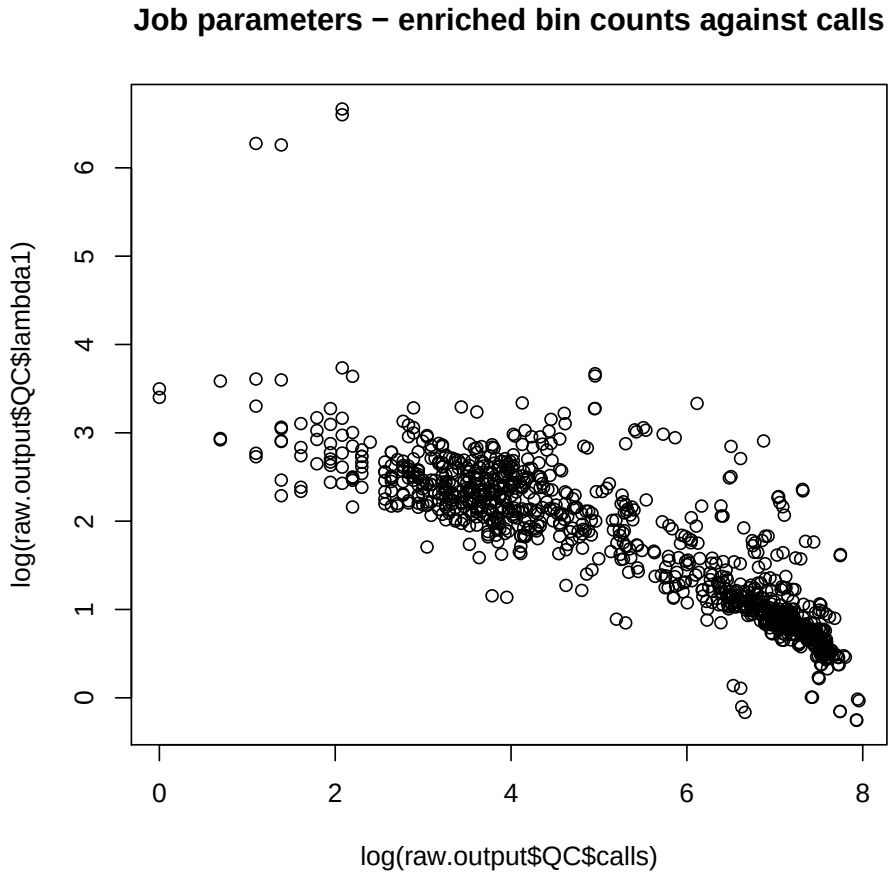

Figure 4: Each point represents a job. On the X axis, we plot the log of the number of bins with $PP > 0.01$ in that job. On the Y axis, we plot the log of the mean number of counts in enriched bins. The plot is on the log scale to aid visualisation of the clusters.

```
> plot(log(raw.output$QC$calls), log(raw.output$QC$lambda1),
+         main = "Job parameters - enriched bin counts against calls")

> plot(log(raw.output$QC$calls), log(raw.output$QC$score),
+         main = "Job parameters - score against calls")
```

The plots generated are given in figures 4 and 5. Two clusters are visible in each, one of which is a cluster of jobs suffering from overfitting.

Some jobs are expected to show no enrichment throughout, such as the centromere, and the location of these regions can be taken into account at a later version of the algorithm. At the moment, our approach is to apply BayesPeak to the whole genome and subsequently to filter out peaks that are allocated within regions of no enrichment.

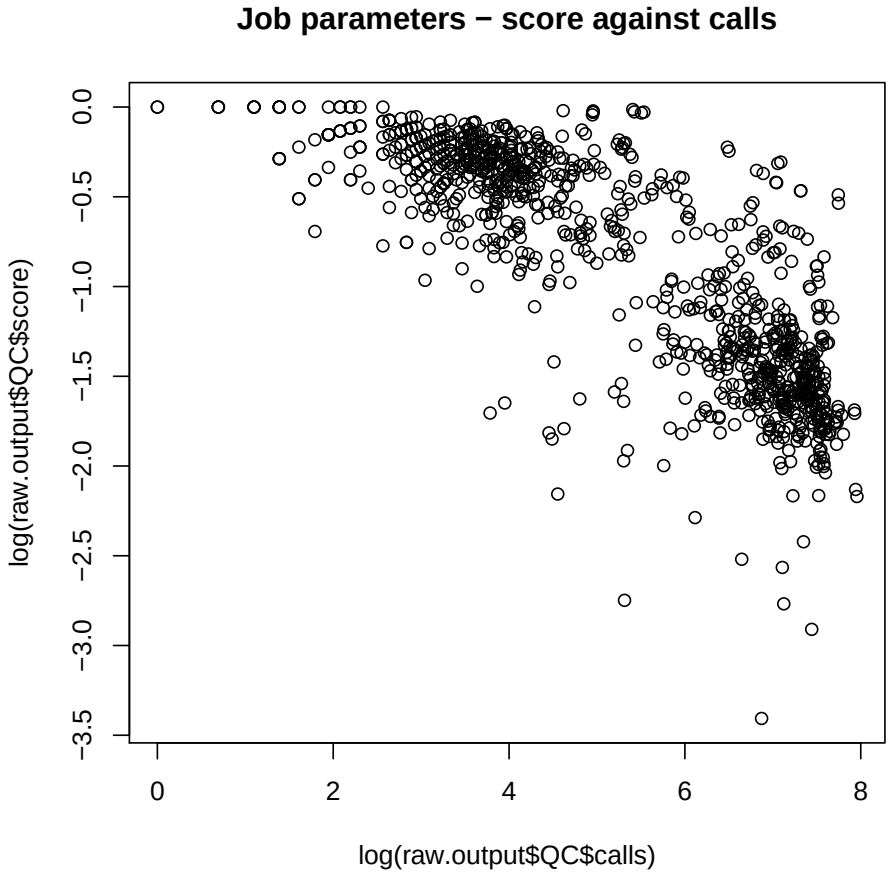

Figure 5: Each point represents a job. On the X axis, we plot the log of the number of bins with $PP > 0.01$ in that job. On the Y axis, we plot the log of the score of that job (i.e. of the bins with $PP > 0.01$, this is the proportion of those bins with $PP > 0.5$).

8.1 Excluding calls from unenriched jobs

We can choose to simply remove all calls from particular jobs. For example, having looked at figure 4, we can specify the overfit cluster by removing all jobs with low counts in their enriched bins - for example, $\log(\lambda_1) < 1.5$:

```
> unreliable.jobs <- log(raw.output$QC$lambda1) < 1.5
> output.sj <- summarise.peaks(raw.output, method = "lowerbound", exclude.jobs = unreliable.jobs)
```

Alternatively, from looking at either figure 4 or figure 5, we could try to specify the overfit cluster better by adding jobs with excessive numbers of calls e.g. $\log(calls) > 5$. This gives us two selection criteria as follows:

```
> unreliable.jobs2 <- log(raw.output$QC$lambda1) < 1.5 | log(raw.output$QC$calls) > 5
> output.sj2 <- summarise.peaks(raw.output, method = "lowerbound", exclude.jobs = unreliable.jobs2)
```

9 Citing BayesPeak

If you use the BayesPeak algorithm, then please cite (C. Spyrou, R. Stark, A.G. Lynch, and S. Tavaré., 2009).

10 Session Info

```
> sessionInfo()
```

```
R version 2.11.1 (2010-05-31)
x86_64-unknown-linux-gnu

locale:
 [1] LC_CTYPE=en_US       LC_NUMERIC=C         LC_TIME=en_US
 [4] LC_COLLATE=en_US     LC_MONETARY=C        LC_MESSAGES=en_US
 [7] LC_PAPER=en_US       LC_NAME=C            LC_ADDRESS=C
[10] LC_TELEPHONE=C       LC_MEASUREMENT=en_US LC_IDENTIFICATION=C

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

other attached packages:
[1] BayesPeak_1.0.1 IRanges_1.6.4
```

11 Acknowledgements

Many thanks to Dr. Duncan Odom's group for permission to use the H3K4me3 data set used in our examples, and to Dr. Jason Carroll's group for permission to use the data set shown in the `raw.output` data file.

References

C. Spyrou, R. Stark, A.G. Lynch, and S. Tavaré. BayesPeak: Bayesian analysis of ChIP-seq data. *BMC Bioinformatics*, 10:299, 2009.

D. Schmidt, M.D. Wilson, C. Spyrou, G.D. Brown, J. Hadfield, and D.T. Odom. ChIP-seq: Using high-throughput sequencing to discover protein-DNA interactions. *Methods*, 48:240–248, 2009.