Using the DMR Scan Package

Christian M Page

30 October 2018

Abstract

A guide to using the DMRScan package for analyzing differantly methylated regions.

Contents

Abstract

Motivation

DNA methylation plays an important role in human health and disease, and methods for the identification of differently methylated regions are of increasing interest. There is currently a lack of statistical methods which properly address multiple testing, i.e. genome-wide significance for differentially methylated regions.

Methods

We introduce a scan statistic (DMRScan), which overcomes these limitations, but don't suffers from the same limitations as alternative R packages, such as *bumphunter*[1] and*DMRcate* [2].

Data Set

In this package, we have included a small bi-sulfite sequncing example data set, which is an extract of chromosome 22.

Work flow and use of DMRScan

Data inputs

Example data

Start by loading the package and have the methylation data ready in the workspace. The minimal data requirements for this package to run, is a list of numerics, with sequentially ordered test statistics, on which the sliding window is used. In this example, we will be using an example data set to generate the needed input.

```
library(DMRScan)
data(DMRScan.methylationData) ## Load methylation data from chromosome 22, with 52018 CpGs measured
data(DMRScan.phenotypes) ## Load phenotype (end-point for methylation data)
```

Test statistics

Note that the DMRScan do not require raw data to be used, only test statistics from a CpG wise model. To generate test statistics for use in DMRScan, we use logistic regression on the example DNA methylation data, and the phenotype data as endpoint, but any other model could as well be used.

```
#observations <- apply(DMRScan.methylationData,1,function(x,y){
#                       summary(glm(y ~ x,
#                       family = binomial(link = "logit")))$coefficients[2,3]},
#                                           y = DMRScan.phenotypes)
```

```
observations <- apply(DMRScan.methylationData,1,function(x,y){
                        summary(lm(x ~ y))$coefficients[2,3]},
                                            y = DMRScan.phenotypes)
head(observations)
## chr22.17061793 chr22.17062497 chr22.17062614 chr22.17063057 chr22.17063105
##     0.02894454     0.65418440     0.85931406    -0.39210304    -0.73461053
## chr22.17063107
##     0.28545561
```

This will return a sequence of test statistics; which is the data that is used further. The output which will be used further is given below:

chr22.17061793	chr22.17062497	chr22.17062614	chr22.17063057	chr22.17063105	chr22.17063107
0.02924113	0.65795471	0.86192751	-0.39548318	-0.73786743	0.28811812

The sliding window function in DMRScan is not dependent on raw methylation data, but rather the test statistic for each genomic position. This allows for much grater flexibility when applying different models to the raw data, and can also be used with meta-analysis values, and not only primary studies.

Clustering of the test statics into “Chunks”

To apply the sliding window on a set of test statistics, we cluster them into chunks with not to much space between the probes. The maximum allowed distance between two probes in the same region is given by `max.gap`, and the minimum number of probes within a cluster is given by `min.cpg`.

The clustering is done automatically by `make.cpg.cluster()` and requires four inputs; - A sequence of test statistics - The corresponding genomic position for the test statistics - The maximum allowed gap (in base pairs) inside a sliding window, i.e. the maximum gap allowed in a cluster - The minimum number of probes in a cluster. This is set to 2 at default.

To identify the coordinate to each test statistic given in the previous table, we split the names inherited from the methylation values into its chromosomal part, and the base pair location.

```
pos <- matrix(as.integer(unlist(strsplit(names(observations),split="chr|[.]"))),
                                                ncol = 3, byrow = TRUE)[,-1]
head(pos)
##      [,1]     [,2]
## [1,]   22 17061793
## [2,]   22 17062497
## [3,]   22 17062614
## [4,]   22 17063057
## [5,]   22 17063105
## [6,]   22 17063107
```

The chromosome number and genomic position is here stored in the same object, a two column matrix. This is not needed, and if more convinient, the two peaces of information can be stored in separate objects.

To generate the clusters, we also need to set the additional clustering parameters:

```
## Minimum number of CpGs in a tested cluster
min.cpg <- 3

## Maxium distance (in base-pairs) within a cluster
## before it is broken up into two seperate cluster
max.gap <- 750
```

This, together with the test statistic vector from earlier, allows us to generate a list of clusters where the sliding window is operated.

```
regions <- makeCpGregions(observations  = observations, chr = pos[,1], pos = pos[,2],
                                                maxGap = 750, minCpG = 3)
```

DMRScan

We are now ready to run the sliding window on the clusters. To run a sliding window, a few things beside the clusters are needed: - The sizes of the windows. This can be either a single window size, or a sequence of windows to be tested. - If a window sequence is given, the multiple testing adjustment will take this into account, and have a more stringent threshold for different window sizes. - The window thresholds need to be calculated. This is dependent on a number of potential settings from the used.

Calculating window thresholds

Three different approaches to estimate the window thresholds is given in the package. The different options are: - A full MCMC model with simulation of null model `model = "mcmc"`, with given correlation structure for the null data, provided by `arima.sim()`. - An faster MCMC called "important sampling", where only a subset of the data is sampled. This is implemented in `model = "sampling"`, and is up to 700 times faster than the MCMC model, and simulation studies indicates that these to options are comparable in performance. - An analytical solution to calculate the window thresholds, based on Siegmund et.al (2012). Given by the model `model = "siegmund"`. Since this is a closed form expression, it is much faster to calculate compared to the two other thresholds. However, this option tends to give more conservative thresholds, with somewhat lower power, but fewer false positive windows. An additional consideration with this method, is that it assumes that the test statistics follows an Ornstein-Uhlenbeck process, which may not always be the case.

To calculate the threshold(s), a few parameters needs to be set; - The method by which to estimate the threshold (see section above). - The window sizes. Can be either a single window size, or a sequence of window sizes. - The total number of observations in the study.

Additionally, for the Monte Carlo options, the number of iterations need to be specified, as well as the correlation structure. We will illustrate using only important sampling (which has a fixed correlation structure on the form of AR(2)) and the closed form expression from Siegmund et.al.

```
window.sizes <- 3:7 ## Number of CpGs in the sliding windows
## (can be either a single number or a sequence)
n.CpG        <- sum(sapply(regions, length)) ## Number of CpGs to be tested
```

```
## Estimate the window threshold, based on the number of CpGs and window sizes
## using important sampling
window.thresholds.importancSampling <- estimateWindowThreshold(nProbe = n.CpG, windowSize = window.sizes,
                                                                method = "sampling", mcmc = 10000)

## Estimating the window threshold using the closed form expression
window.thresholds.siegmund <- estimateWindowThreshold(nProbe = n.CpG, windowSize = window.sizes,
                                                      method = "siegmund")
```

Identifying Differentially Methylated Regions

We now have all the data and parameters needed to run the `dmrscan()` function. First using a window threshold estimated with *importanc sampling*, we have

```
window.thresholds.importancSampling <- estimateWindowThreshold(nProbe = n.CpG, windowSize = window.sizes,
method = "sampling", mcmc = 10000)
dmrscan.results   <- dmrscan(observations = regions, windowSize = window.sizes,
                                    windowThreshold = window.thresholds.importancSampling)
## Print the result
print(dmrscan.results)
## GRanges object with 6 ranges and 3 metadata columns:
##       seqnames            ranges strand |    no.cpg                pVal
##          <Rle>         <IRanges>  <Rle> | <integer>            <numeric>
##   [1]       22 22874560-22874588      * |         4 0.000223926552090914
##   [2]       22 23801059-23801103      * |         5 0.000569146653231074
##   [3]       22 23801271-23801333      * |         7 0.000886375935359869
##   [4]       22 23801377-23801402      * |         3 4.66513650189405e-05
##   [5]       22 23801581-23801591      * |         3 3.73210920151524e-05
##   [6]       22 29704113-29704128      * |         4 0.000289238463117431
##                  tVal
##             <numeric>
##   [1] 2.62950365436819
##   [2] 2.34921457079467
##   [3] 2.20187418841811
##   [4] 2.99539831121707
##   [5] 3.32117224153482
##   [6]  2.5845665979541
##   -------
##   seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

This will give the following result, with two genome wide significant regions.

Genomic Coordinate	#CpG	Empirical P value
Chr22:23801581-23801591	3	3.73210920151524e-05
Chr22:23801271-23801333	7	0.000755752113306835
Chr22:23801059-23801111	7	0.000951687846386385

When using a more stringent cut-off, generated by using the "*siegmund*" option in `dmrscan()`, we get no significant regions on the same data set. Exemplified by the syntax below:

```
dmrscan.results   <- dmrscan(observations = regions, windowSize = window.sizes,
                             windowThreshold = window.thresholds.siegmund)
## Print the result
print(dmrscan.results)
## GRanges object with 1 range and 3 metadata columns:
##       seqnames            ranges strand |    no.cpg                pVal
##          <Rle>         <IRanges>  <Rle> | <integer>            <numeric>
##   [1]       22 23801581-23801591      * |         3 3.73210920151524e-05
##                   tVal
##              <numeric>
##   [1] 3.32117224153482
##   -------
##   seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

Estimating window thresholds with an ARIMA model using MCMC

We can also use an Monte Carlo approach to estimate the window thresholds, in order to model complex or non-standard correlation structures. We use the argument "submethod" to set the function calls to the different sampling functions (e.g. ar(), ma(), arima()), and pases ... to the models argument in these functions.

```
# Not run due to time constraints.
# window.threshold.mcmc <- estimateWindowThreshold(nProbe = n.CpG, windowSize = window.sizes,
#                               method = "mcmc", mcmc = 1000, nCPU = 1, submethod = "arima",
#                               model = list(ar = c(0.1,0.03), ma = c(0.04), order = c(2,0,1)))
#
# dmrscan.results  <- dmrscan(observations = regions, windowSize = window.sizes,
#                               windowThreshold = window.thresholds.mcmc)
## Print the result
#print(dmrscan.results)
```

References

[1] Jaffe AE, Murakami P, Lee H, Leek JT, Fallin DM, Feinberg AP and Irizarry RA (2012). "Bump hunting to identify differentially methylated regions in epigenetic epidemiology studies." *International journal of epidemiology*, 41(1), pp. 200–209. doi: 10.1093/ije/dyr238.

[2] Peters TJ, Buckley MJ, Statham AL, Pidsley R, Samaras K, Lord RV, Clark SJ and Molloy PL (2015). "De novo identification of differentially methylated regions in the human genome." *Epigenetics & Chromatin*, 8, pp. 6. http://www.epigeneticsandchromatin.com/content/8/1/6.

```
sessionInfo()
## R version 3.5.1 Patched (2018-07-12 r74967)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 16.04.5 LTS
##
## Matrix products: default
## BLAS: /home/biocbuild/bbs-3.8-bioc/R/lib/libRblas.so
```

```
## LAPACK: /home/biocbuild/bbs-3.8-bioc/R/lib/libRlapack.so
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=C
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] DMRScan_1.8.0    BiocStyle_2.10.0
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.19           XVector_0.22.0         knitr_1.20
##  [4] magrittr_1.5           MASS_7.3-51            zlibbioc_1.28.0
##  [7] GenomicRanges_1.34.0   BiocGenerics_0.28.0    IRanges_2.16.0
## [10] RcppRoll_0.3.0         lattice_0.20-35        stringr_1.3.1
## [13] GenomeInfoDb_1.18.0    tools_3.5.1            grid_3.5.1
## [16] parallel_3.5.1         xfun_0.4               htmltools_0.3.6
## [19] yaml_2.2.0             rprojroot_1.3-2        digest_0.6.18
## [22] bookdown_0.7           Matrix_1.2-14          GenomeInfoDbData_1.2.0
## [25] BiocManager_1.30.3     codetools_0.2-15       S4Vectors_0.20.0
## [28] bitops_1.0-6           RCurl_1.95-4.11        evaluate_0.12
## [31] rmarkdown_1.10         stringi_1.2.4          compiler_3.5.1
## [34] backports_1.1.2        stats4_3.5.1           mvtnorm_1.0-8
```

```
R version 3.4.2 (2017-09-28)
Platform: x86_64-apple-darwin15.6.0 (64-bit)
Running under: macOS Sierra 10.12.6

Matrix products: default
BLAS: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRlapack.dylib

locale:
[1] C

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

other attached packages:
[1] DMRScan_0.01.02

loaded via a namespace (and not attached):
 [1] Rcpp_0.12.13            mvtnorm_1.0-6           lattice_0.20-35
 [4] IRanges_2.10.5          RcppRoll_0.2.2          bitops_1.0-6
 [7] MASS_7.3-47             GenomeInfoDb_1.12.3     grid_3.4.2
```

```
[10] stats4_3.4.2           zlibbioc_1.22.0         XVector_0.16.0
[13] S4Vectors_0.14.7       Matrix_1.2-11           RCurl_1.95-4.8
[16] parallel_3.4.2         compiler_3.4.2          BiocGenerics_0.22.1
[19] GenomicRanges_1.28.6   GenomeInfoDbData_0.99.0
```

End of vignette