Methylaction: Detecting differentially methylated regions (DMRs) that distinguish disease subtypes

Jeff Bhasin

2015-04-28

Contents

Purpose

This how-to will demonstrate the use of Methylaction to detect differentially methylated regions (DMRs) among three groups using data from MBD-isolated Genome Sequencing (MiGS). While Methylaction is designed for genome-wide analysis, this example data is only for a subset of the genome and a subset of samples so the example can be worked through quickly. Please refer to the function documentation for advanced options.

Prerequisites

- R Installed (download from http://cran.r-project.org/)
- A linux server or workstation

RAM and CPU requirements will depend on the depth of the sequencing and the number of samples/groups. We recommend very high performance machines. As a reference, we used a linux server with 20 cores of

2.80GHz CPUs and 64GB of RAM for the whole genome analysis of 22 samples across 3 groups. A high performance computing cluster (HPC) was used to obtain 1,000s of permutations.

For the purposes of the example, less CPU and RAM are required.

Installation

First, start R and install pre-requisite packages from Bioconductor:

```
source("http://bioconductor.org/biocLite.R")
biocLite(c("GenomicRanges","IRanges","devtools","DESeq","GenomicAlignments","Repitools","Rsubread","ggb
```

Then, install both Goldmine and Methylaction from GitHub. Be sure to accept installation of any additional pre-requisite packages from CRAN.

```
library(devtools)
install_github("jeffbhasin/goldmine")
install_github("jeffbhasin/methylaction")
```

Example Data

Please obtain the "methylaction_demo.zip" file and extract it. This contains all the example input data needed to complete this how-to.

Preprocessing

First, load the methylaction R package into a new session of R.

```
library(methylaction)
```

Describing the Experimental Design

Unique sample identifiers, BAM files, group assignments, and any sample covariates are defined in a CSV file. Please see "input/samples.csv" for an example of the format. Note that groups must be ordered in the order you want the groups to appear in the output results. This is important, because in the output, patterns between the groups will be coded using sequences of binary digits, where each digit represents a group, and this order is the order the groups are encountered in the sample CSV file. Optionally, a column called "color" can be provided that defines colors for each groups. These will be used in certain plotting and reporting functions. If this column is omitted, colors are automatically assigned using RColorBrewer.

The sample CSV file can be read into R using the readSampleInfo() command:

```
samp <- readSampleInfo("input/samples.csv")
```

The command will output confirmation of group sample sizes and group order.

Loading Reads and Generating Count Tables

To save time, all read alignments are read from the BAM files into an RData workspace that is saved to disk, and this prevents you from having to re-read this data from BAM any time you want to re-run methylaction. The getReads() function does this. The initial stage of the program works with read counts in 50bp windows, which can also be computed once, and do not have to be re-run for any re-run of the methylaction() command. The getCounts() function does this.

There are two other variables specific to the experiment that will be needed in the next step. These are the window size (we recommend 50bp) and the fragment size which was selected for in your sequencing protocol. This fragment size is not the read length – it is the average size of a fragment in the sequencing library that was prepared. Often this number is available from BioAnayzer results, and should be known to whomever did the sequencing library preparation for your study.

We recommend saving all of the above into a single preprocessing RData, which can be loaded prior to running the DMR detection step described next.

First, define some variables that will be needed for the preprocessing functions (change to suit your experiment, especially fragsize which is the fragment length selected for during sequencing library preparation):

```
chrs <- "chr22"
fragsize <- 120
winsize <- 50
ncore <- 1
bsgenome <- Hsapiens
```

Then, read in alignments and generate binned count tables (these steps may be memory, disk, and CPU intensive):

```
reads <- getReads(samp=samp, chrs=chrs, fragsize=fragsize, ncore=ncore)
counts <- getCounts(samp=samp, reads=reads, chrs=chrs, winsize=winsize, ncore=ncore)
```

Finally, save all of the above to an RData file:

```
save(samp, reads, counts, winsize, fragsize, chrs, file="output/prepro.rd",compress=T)
```

For future runs of DMR detection, the saved RData can be loaded rather than spending time re-preprocessing the data.

Differentially Methylated Region (DMR) Detection

With all the preprocessing completed, differentially methylated regions (DMRs) can be detected using a call to the methylaction() function. There are many options to this function that will affect the DMR detection. Here we have used recommended defaults. See the function documentation for more details. This function performs multiple steps, which are both CPU, RAM, and disk intensive when run on larger data sets.

First, set the number of cores to use based on your hardware (we recommend reducing this number if there are memory issues):

```
ncore <- 1
```

Then, run methylaction() to call DMRs:

```
ma <- methylaction(samp=samp, counts=counts, reads=reads, winsize=winsize, ncore=ncore)
```

Finally, save the results object to disk:

```
save(ma,file="output/ma.rd", compress=T)
```

Interpreting the results

The function methylaction() is designed to output a great deal of information about the internals of the DMR calling in order to facilitate comparisons between different settings and to prevent needing to re-run the command on large datasets to view intermediate states. The output object is a list. For a list of all detected DMRs, look at ma$dmr. Each DMR is assigned a pattern code, where each digit represents a group. The pattern indicates the differential methylation status between the groups (see table for clarification). To see what options the list was created using, see the ma$args object. If you want to access data from any internal steps of the function, see the objects nested under ma$data.

Vizualization of DMRs

DMRs can be visualized genome-wide via a heatmap or karyogram.

```
maHeatmap(ma,file="heat.png")
maKaryogram(ma,file="karyo.png")
```

Permutation and Bootstrap Testing

Because of the two stage testing approach, type I error rates are inflated with this method. To determine if this level is acceptable, we have implemented a permutation approach. This establishes a FDR for each pattern of DMR between each group. If these FDRs are too high, they can be recalculated at lower p-value thresholds until they reach acceptable levels. Then, DMRs cut at this p-value can be used as the definitive list for the study.

Bootstraps can be enabled by adding the "nperms" option to methylaction(). The resulting output list will then have an "fdr" object that reports false discovery rates (FDRs).

```
ma <- methylaction(samp=samp, counts=counts, reads=reads, winsize=winsize, perm.boot=T, nperms=3, ncore=
```

If "perm.boot" is set to be FALSE, then regular permutations (sampling without replacement) are performed rather than bootstrapping (sampling with replacement).

See the maPerm(), maPermMerge(), and maPermFdr() for manual methods to run permutations. These are useful for spreading permutations across multiple machines or in a high performance computing (HPC) environment.