DNA methylation tutorial: Data Analysis

by
Roseric Azondekon, PhD
University of Wisconsin Milwaukee

June 10, 2019

Background

In a previous tutorial, we showed you how to download and process Bisulfite-seq DNA methylation FASTQ files for read alignment on a reference sequence. In this tutorial, we show you how to run DNA methylation analysis using the `methylKit` package in `R`.

We set our working directory to the `tuto` folder created in our first tutorial.

```
In [ ]: setwd('./tuto')
```

Now, let's install all the required packages for this tutorial.

```
In [ ]: # Indicate package repositories to R...
        repositories <- c("https://cloud.r-project.org",
                          "https://bioconductor.org/packages/3.7/bioc",
                          "https://bioconductor.org/packages/3.7/data/annotation",
                          "https://bioconductor.org/packages/3.7/data/experiment",
                          "https://www.stats.ox.ac.uk/pub/RWin",
                          "http://www.omegahat.net/R",
                          "https://R-Forge.R-project.org",
                          "https://www.rforge.net",
                          "https://cloud.r-project.org",
                          "http://www.bioconductor.org",
                          "http://www.stats.ox.ac.uk/pub/RWin")

        # Package list to download
        packages <- c("BiocGenerics","Biobase","S4Vectors","IRanges",
                      "GenomicRanges","GenomeInfoDb","AnnotationDbi",
                      "genomation", "fastseg", "methylKit")

        # Install and load missing packages
        new.packages <- packages[!(packages %in% installed.packages()[,"Package"])]
```

```
if(length(new.packages)){
    install.packages(new.packages, repos = repositories)
}

lapply(packages, require, character.only = TRUE)
```

1 Obtaining methylation percentage from sorted Bismark alignments

We read in the methylation calls directly from the `SAM` files obtained from the last tutoral. The SAM files must be sorted and be generated from Bismark aligner. For that purpose, we use the `processBismarkAln` function from the `methylKit` package as described below:

```
In [ ]: # Find all sorted SAM files
        file_loc <- file.path(getwd(),'alignment_Bismark')
        file.list <- as.list(list.files(file_loc, "\\.sort.sam$",
                                        full.names = TRUE, recursive = TRUE))
        file.list
```

Now, let's read in the methylation data from the `SAM` files (*this has to be run once*).
This may take a long time!

```
In [ ]: # Read the methylation data into an object (Has to be run once)
        metyl.obj <- processBismarkAln(file.list,
                                       sample.id = list("normal1","normal2",
                                                        "cancer1","cancer2"),
                                       treatment = c(0,0,1,1),
                                       assembly = "hg38",
                                       read.context = "CpG",
                                       save.folder = file.path(getwd(),'methylation_calls'))
```

```
In [ ]: head(metyl.obj,3)
```

When already computed, the set of methylation call files can be imported using the `methRead` function provided by the `methylKit` package.

```
In [ ]: # This code may be run if the methylation call files already exist
        file_loc2 <- file.path(getwd(),'methylation_calls')
        file.list2 <- as.list(list.files(file_loc2, "\\CpG.txt$",
                                         full.names = TRUE,
                                         recursive = TRUE))
        file.list2
```

```
In [ ]: methyl.obj2 <- methRead(file.list2,
                                sample.id = list("cancer1","cancer2",
                                                 "normal1","normal2"),
                                treatment=c(1,1,0,0),
                                assembly = "hg38")
```

```
In [ ]: head(methyl.obj2,3)
```

2 Quality check and basic features of the data

Let's check the basic stats about the methylation data such as coverage and percent methylation.

```
In [ ]: # Descriptive statistics on the samples
        getMethylationStats(metyl.obj[[1]], plot = F, both.strands = F) # normal1
        getMethylationStats(metyl.obj[[2]], plot = F, both.strands = F) # normal2
        getMethylationStats(metyl.obj[[3]], plot = F, both.strands = F) # cancer1
        getMethylationStats(metyl.obj[[4]], plot = F, both.strands = F) # cancer2
```

```
In [ ]: # Histogram of % CpG methylation
        getMethylationStats(metyl.obj[[1]], plot = T, both.strands = F) # normal1
        getMethylationStats(metyl.obj[[2]], plot = T, both.strands = F) # normal2
        getMethylationStats(metyl.obj[[3]], plot = T, both.strands = F) # cancer1
        getMethylationStats(metyl.obj[[4]], plot = T, both.strands = F) # cancer2
```

On the histograms above, the numbers on bars represent the percentage of locations that are contained in that bin. Percent methylation histogram are expected to have two peaks on both ends.

Alternatively, the histogram of the read coverage per base information can be plotted as well. Experiments that are highly suffering from PCR duplication bias are expected to have a secondary peak towards the right hand side of the histogram.

```
In [ ]: # Histogram of CpG coverage
        getCoverageStats(metyl.obj[[1]], plot = T, both.strands = F) # normal1
        getCoverageStats(metyl.obj[[2]], plot = T, both.strands = F) # normal2
        getCoverageStats(metyl.obj[[3]], plot = T, both.strands = F) # cancer1
        getCoverageStats(metyl.obj[[4]], plot = T, both.strands = F) # cancer2
```

3 Filtering samples based on read coverage

It is a good practice to filter samples based on coverage, and discard bases that have coverage below 10X bases that have more than 99.9th percentile of coverage in each sample. This can be achieved with the following code:

```
In [ ]: filtered.metyl.obj <- filterByCoverage(metyl.obj,
                                                 lo.count = 10,
                                                 lo.perc = NULL,
                                                 hi.count = NULL,
                                                 hi.perc = 99.9)
```

Let's assess once again the basic stats about the methylation data such as coverage and percent methylation.

```
In [ ]: # Histogram of % CpG methylation
        getMethylationStats(filtered.metyl.obj[[1]], plot = T, both.strands = F) # normal1
        getMethylationStats(filtered.metyl.obj[[2]], plot = T, both.strands = F) # normal2
        getMethylationStats(filtered.metyl.obj[[3]], plot = T, both.strands = F) # cancer1
        getMethylationStats(filtered.metyl.obj[[4]], plot = T, both.strands = F) # cancer2
```

```
In [ ]: # Histogram of CpG coverage
        getCoverageStats(filtered.metyl.obj[[1]], plot = T, both.strands = F) # normal1
        getCoverageStats(filtered.metyl.obj[[2]], plot = T, both.strands = F) # normal2
        getCoverageStats(filtered.metyl.obj[[3]], plot = T, both.strands = F) # cancer1
        getCoverageStats(filtered.metyl.obj[[4]], plot = T, both.strands = F) # cancer2
```

4 Sample correlation

To conduct sample correlation, we will need to merge all samples to one object for base-pair locations that are covered in all samples.

```
In [ ]: # Merging all samples
        merged.obj = unite(filtered.metyl.obj, destrand = FALSE)
        # Taking a glance at the data...
        head(merged.obj,3)
```

The sample correlation is computed using the `getCorrelation` function availble in `methylKit`.

```
In [ ]: # Sample correlation
        getCorrelation(merged.obj, plot = T)
```

5 Clustering samples

The `clusterSamples` function in `methylKit` can be used to perform the hierarchical clustering of the samples based on their methylation profiles.

```
In [ ]: clusterSamples(merged.obj, dist = "correlation", method = "ward", plot=T)
```

Principal Component Analysis (PCA) is another available method to cluster the samples. We perform PCA using the the `PCASamples` function, then plot the Screeplot.

```
In [ ]: # Screeplot (PCA analysis)
        PCASamples(merged.obj, screeplot = TRUE)
```

Here, we plot PC1 (principal component 1) and PC2 (principal component 2) axis and a scatter plot of our samples on those axis which reveals how they cluster.

```
In [ ]: # Scatterplot (PCA analysis)
        PCASamples(merged.obj)
```

6 Getting differentially methylated bases

The function `calculateDiffMeth` is the main function to calculate differential methylation.

```
In [ ]: # Finding differentially methylated bases or
        # regions (using 8 cores for faster calculations)
        methyl.diff <- calculateDiffMeth(merged.obj, num.cores = 8)
```

Following bit selects the bases that have q-value < 0.01 and percent methylation difference larger than 25%.

```
In [ ]: # get hyper methylated bases
        myDiff25p.hyper <- getMethylDiff(methyl.diff,
                                         difference = 25,
                                         qvalue = 0.01,
                                         type = "hyper")

        # get hypo methylated bases
        myDiff25p.hypo <- getMethylDiff(methyl.diff,
                                        difference = 25,
                                        qvalue = 0.01,
                                        type = "hypo")

        # get all differentially methylated bases
        myDiff25p <- getMethylDiff(methyl.diff,
                                   difference = 25,
                                   qvalue = 0.01)
```

The package `methylKit` can summarize methylation information over tiling windows or over a set of predefined regions (promoters, CpG islands, introns, etc.) rather than doing base-pair resolution analysis.

```
In [ ]: tiles <- tileMethylCounts(merged.obj, win.size = 1000, step.size = 1000)
        head(tiles, 3)
```

7 Differential methylation events per chromosome

We can also visualize the distribution of hypo/hyper-methylated bases/regions per chromosome using the `diffMethPerChr` function.

```
In [ ]: # Return a list having per chromosome
        # differentially methylation events will be returned
        diffMethPerChr(methyl.diff,
                       plot = FALSE,
                       qvalue.cutoff = 0.01,
                       meth.cutoff = 25)
```

```
In [ ]: # visualize the distribution of hypo/hyper-methylated
        # bases/regions per chromosome
        diffMethPerChr(methyl.diff,
                       plot = T,
                       qvalue.cutoff = 0.01,
                       meth.cutoff = 25)
```

8 Annotating differential methylation events

We can annotate our differentially methylated regions/bases based on gene annotation. We need to read the gene annotation from a bed file and annotate our differentially methylated regions with that information. Similar gene annotation can be created using `GenomicFeatures` package available from Bioconductor.

Let's first download an annotation `.bed` file in the `annotation` folder created in our first tutorial on DNA methylation. We get ours from sourceforge.net.

```
In [ ]: url <- "https://sourceforge.net/projects/rseqc/files/BED/Human_Homo_sapiens/"
        system(paste0("wget -P annotation ",url,"hg38_RefSeq.bed.gz"))
```

```
In [ ]: gene.obj <- readTranscriptFeatures(file.path(getwd(),
                                                     "annotation",
                                                     "hg38_RefSeq.bed.gz"))
```

```
In [ ]: annotateWithGeneParts(as(myDiff25p,"GRanges"),gene.obj)
```

Similarly, we can read the CpG island annotation and annotate our differentially methylated bases/regions with them.

```
In [ ]: # read the shores and flanking regions and name the flanks as shores
        # and CpG islands as CpGi
        cpg.obj <- readFeatureFlank(file.path(getwd(),"annotation","hg38_RefSeq.bed.gz"),
                                    feature.flank.name=c("CpGi","shores"))
```

```
In [ ]: # convert methylDiff object to GRanges and annotate
        diffCpGann <- annotateWithFeatureFlank(as(myDiff25p,"GRanges"),
                                               cpg.obj$CpGi,
                                               cpg.obj$shores,
                                               feature.name="CpGi",
                                               flank.name="shores")
```

```
In [ ]: diffCpGann
```

9 Regional analysis

We now summarize methylation information over a set of defined regions such as promoters or CpG islands with the `regionCounts` function.

```
In [ ]: promoters <- regionCounts(metyl.obj,gene.obj$promoters)
        head(promoters[[1]])
```

10 Getting the distance to TSS and nearest gene name

After getting the annotation of differentially methylated regions, we can get the distance to TSS and nearest gene name using the `getAssociationWithTSS` function from genomation package.

```
In [ ]: diffAnn <- annotateWithGeneParts(as(myDiff25p, "GRanges"), gene.obj)

        # target.row is the row number in myDiff25p
        head(getAssociationWithTSS(diffAnn),3)
```

Getting the percentage/number of differentially methylated regions that overlap with intron/exon/promoters:

```
In [ ]: genomation::getTargetAnnotationStats(diffAnn,
                                             percentage=TRUE,
                                             precedence=TRUE)
```

11 Working with annotated methylation events

We can also plot the percentage of differentially methylated bases overlapping with exon/intron/promoters

```
In [ ]: genomation::plotTargetAnnotation(diffAnn,
                                         precedence = TRUE,
                                         main = "Differential methylation annotation")
```

It is also possible to plot the CpG island annotation showing the percentage of differentially methylated bases that are on CpG islands, CpG island shores and other regions.

```
In [ ]: genomation::plotTargetAnnotation(diffCpGann,
                                         col = c("green", "gray", "white"),
                                         main = "differential methylation annotation")
```