Lab 6: Minfi

Jean-Philippe Fortin

May 2, 2014

1 Introduction

The goal of today's tutorial is to present a standard analysis workflow of 450K data with the package minfi, incorporating the most recent functions added to the package.

We will start from the very beginning by reading input raw data (IDAT files) from an example dataset, and ending with a list of candidate genes for differential methylation. Among others, we will cover quality control assessments, different within-array and between-array normalizations, SNPs removal, sex prediction, differentially methylated positions (DMPs) analysis and bump hunting for differentially methylated regions (DMRs).

If time permits, we will introduce a complementary visualization tool package, shinyMethyl, that allows interactive quality control assessment and principal component analysis (PCA). You can download the package online from shinymethyl.com and try it online.

450k Array design and terminology

In this section, we introduce briefly the 450K array as well as the terminology used throughout the minfi package. Each sample is measured on a single array, in two different color channels (red and green). As the name of the platform indicates, each array measures more than 450,000 CpG positions. For each CpG, we have two measurements: a methylated intensity and an

unmethylated intensity. Depending on the probe design, the signals are reported in different colors:

For **Type I** design, both signals are measured in the same color: one probe for the methylated signal and one probe for the unmethylated signal.

For **Type II** design, only one probe is used. The **Green** intensity measures the methylated signal, and the **Red** intensity measures the unmethylated signal.

Some definitions

Two commonly measures are used to report the methylation levels: Beta values and M values.

Beta value:

$$\beta = \frac{M}{M + U + 100}$$

where M and U denote the methylated and unmethylated signals respectively.

MValue:

$$Mval = \log\left(\frac{M}{U}\right)$$

DMP: Differentially methylated position: single genomic position that has a different methylated level in two different groups of samples (or conditions)

DMR: Differentially methylated region: when consecutive genomic locations are differentially methylated in the same direction.

Array: One sample

Slide: Physical slide containing 12 arrays (6×2 grid)

Plate: Physical plate containing at most 8 slides (96 arrays). For this tutorial, we use **batch** and plate interchangeably.

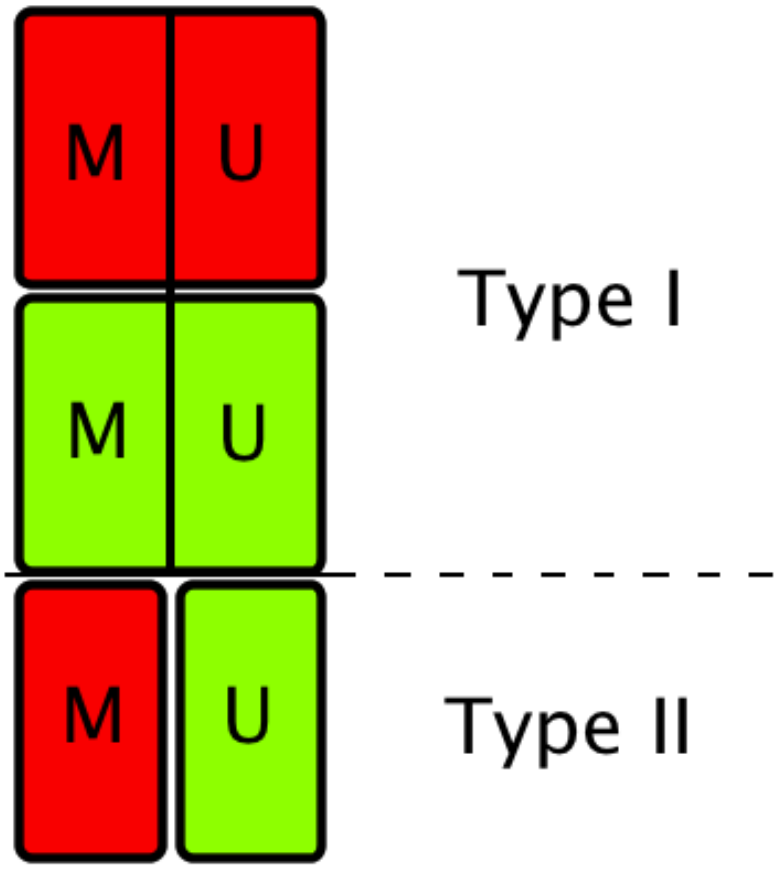

Figure 1: **Probe design of the 450k array** For **Type I** design, both signals are measured in the same color: one probe for the methylated signal and one probe for the unmethylated signal. For **Type II** design, only one probe is used. The **Green** intensity measures the methylated signal, and the **Red** intensity measures the unmethylated signal.

2 Reading Data

The starting point of minfi is reading the .IDAT files with the built-in function `read.450k.exp`. Several options are available: the user can specify the sample filenames to be read in along with the directory path, or can specify the directory that contains the files. In the latter case, all the files with the extension .IDAT located in the directory will be loaded into R. The user can also read in a sample sheet, and then use the sample sheet to load the data into a `RGChannelSet`. For more information, see the `minfi` vignette. Here, we will load the dataset containing 6 samples from the `minfiData` package using the sample sheet provided within the package:

```
> require(minfi)
> require(minfiData)

> baseDir <- system.file("extdata",package="minfiData")
> targets <- read.450k.sheet(baseDir)
> targets
> RGSet <- read.450k.exp(base = baseDir, targets = targets)
```

The class of RGSet is a `RGChannelSet` object. This is the initial object of a minfi analysis that contains the raw intensities in the green and red channels. Note that this object contains the intensities of the internal control probes as well. Because we read the data from a data sheet experiment, the phenotype data is also stored in the `RGChannelSet` and can be accessed via the accessor command `pData`:

```
> phenoData <- pData(RGSet)
> phenoData[,1:6]
```

The `RGChannelSet` stores also a manifest object that contains the probe design information of the array:

```
> manifest <- getManifest(RGSet)
> manifest
> head(getProbeInfo(manifest))
```

For instance, one can extract the probe names for Type I probes as follows:

```
> typeIProbes <- getProbeInfo(manifest, type = "I")$Name
> head(typeIProbes)
```

The 450K array contains also 65 SNP probes that do not interrogate methylation. They can be used as control probes to check whether or not samples have been mixed up:

```
> snpProbesI <- getProbeInfo(manifest, type = "SnpI")$Name
> snpProbesII <- getProbeInfo(manifest, type = "SnpII")$Name
> head(snpProbesI)
```

3 Quality control

minfi provides several functions and diagnostic plots to assess quality of the methylation samples. As a starting point, we suggest to look at the function `detectionP()` which identifies failed positions defined as both the methylated and unmethylated channel reporting background signal levels:

```
> detP <- detectionP(RGSet)
> failed <- detP > 0.01
> head(failed, n=3)
```

To compute the fraction of failed positions per sample:

```
> colMeans(failed)
```

and to compute for instance how many positions failed in $> 50\%$ of the samples:

```
> sum(rowMeans(failed)>0.5)
```

QC plot

minfi provides a simple quality control plot that uses the log median intensity in both the methylated (M) and unmethylated (U) channels. When plotting these two medians against each other, it has been observed that good samples cluster together, while failed samples tend to separate and have lower median intensities. We need need to convert the `RGChannelSet` to an object containing the methylated and unmethylated signals using the function

preprocessRaw. It takes as input a RGChannelSet and converts the red and green intensities to methylated and unmethylated signals according to the probe design stored in the manifest object, and returns the converted signals in a new object of class MethylSet. Notice that no normalization has been done so far.

```
> MSet <- preprocessRaw(RGSet)
> MSet
```

The accessors getMeth and getUnmeth can be used to access the methylated and unmethylated intensities:

```
> head(getMeth(MSet)[,1:3])
> head(getUnmeth(MSet)[,1:3])
```

We will talk more about the MethylSet later. The functions getQC and plotQC are designed to extract the quality control information from the MethylSet:

```
> qc <- getQC(MSet)
> head(qc)
> plotQC(qc)
```

To further explore the quality of the samples, it is useful to look at the Beta value densities of the samples, with the option to color the densities by group:

```
> densityPlot(MSet, sampGroups = phenoData$Sample_Group, main= "Beta", xlab = "B
```

or density bean plots:

```
> densityBeanPlot(MSet, sampGroups = phenoData$Sample_Group)
```

Finally, the 450k array contains several internal control probes that can be used to assess the quality control of different sample preparation steps (bisulfite conversion, hybridization, etc.). The values of these control probes are stored in the initial RGChannelSet and can be plotted by using the function controlStripPlot and by specifying the control probe type:

```
> controlStripPlot(RGSet, controls="BISULFITE CONVERSION II")
```

All the plots above can be exported into a pdf file in one step using the function qcReport:

```
> qcReport(RGSet, pdf= "qcReport.pdf")
```

When the number of samples becomes large, it becomes difficult to assess quality control using the generated plots above.

4 MethylSet and RatioSet

As said before, the `MethylSet` contains the methylated and unmethylated signals, but no methylation level statistic such as Beta values of M values. We can compute the Beta values or M values from a `MethylSet` using the commands `getBeta` and `getM`:

```
> getBeta(MSet, type = "Illumina")[1:4,1:3]
> getM(MSet)[1:4,1:3]
```

Note that the option `type = "Illumina"` means that a constant of 100 is added in the denominator. Similarly, as seen before, `getMeth` and `getUnmeth` return methylated and unmethylated intensities respectively .

The `RatioSet` object class is designed to store Beta values and/or M values instead of the methylated and unmethylated signals. An optional copy number matrix, `CN`, which is the sum of the methylated and unmethylated signals, can be also stored. Mapping a `MethylSet` to a `RatioSet` is irreversible, i.e. one cannot technically retrieve the methylated and unmethylated signals from a `RatioSet`. A `RatioSet` can be created with the function `ratioConvert`:

```
> ratioSet <- ratioConvert(MSet, what = "both", keepCN = TRUE)
> ratioSet
```

The functions `getBeta`, `getM` and `getCN` return respectively the Beta value matrix, M value matrix and a the Copy Number matrix.

5 SNPs

6 Map to Genome

The `RatioSet` object is an object containing the final Beta values and M values for further analysis. The function `mapToGenome` applied to a `RatioSet` object will add genomic coordinates to each probe together with some additional annotation information. The output object is a `GenomicRatioSet` (class holding M or/and Beta values together with associated genomic coordinates). It is possible to merge the manifest object with the genomic locations by setting the option `mergeManifest` to `TRUE`.

```
> gRatioSet <- mapToGenome(ratioSet, mergeManifest = TRUE)
> gRatioSet
> showClass("GenomicRatioSet")
```

Note that the `GenomicRatioSet` extends the class `SummarizedExperiment`. Here are the main accessors functions to access the data:

```
> getBeta(gRatioSet)
> getM(gRatioSet)
> getCN(gRatioSet)
```

```
> sampleNames <- sampleNames(gRatioSet)
> probeNames <- featureNames(gRatioSet)
> pheno <- pData(gRatioSet)
```

The accessor `rowData` will extract a `GenomicRanges` object from the `GenomicRatioSet`:

```
> gRanges <- rowData(gRatioSet)
> head(gRanges, n= 3)
```

To get the full annotation available, one can use the command `getAnnotation`:

```
> annotation <- getAnnotation(gRatioSet)
> names(annotation)
```

7 Preprocessing and normalization

So far, we have processed the data with no normalization at all. Different normalization procedures are available in `minfi`.

preprocessRaw

As seen before, it converts a `RGChannelSet` to a `MethylSet` by converting the Red and Green channels into a matrix of methylated signals and a matrix of unmethylated signals. No normalization is performed.

Input: `RGChannelSet`
Output: `MethylSet`

preprocessIllumina

Convert a `RGChannelSet` to a `MethylSet` by implementing the preprocessing choices as available in Genome Studio: background subtraction and control normalization. Both of them are optional and turning them off is equivalent to raw preprocessing (`preprocessRaw`):

```
> MSet.illumina <- preprocessIllumina(RGSet, bg.correct = TRUE,
+                                     normalize = "controls")
```

Input: `RGChannelSet`
Output: `MethylSet`

preprocessSWAN

Perform Subset-quantile within array normalization (SWAN) [4], a within-array normalization correction for the technical differences between the Type I and Type II array designs. The algorithm matches the Beta-value distributions of the Type I and Type II probes by applying a within-array quantile normalization separately for different subsets of probes (divided by CpG content). The input of SWAN is a `MethylSet`, and the function returns a `MethylSet` as well. If an `RGChannelSet` is provided instead, the function will first call `preprocessRaw` on the `RGChannelSet`, and then apply the SWAN normalization. We recommend setting a seed (using `set.seed`) before using `preprocessSWAN` to ensure that the normalized intensities will be reproducible.

```
> MSet.swan <- preprocessSWAN(RGSet)
```

Input: `RGChannelSet` or `MethylSet`
Output: `MethylSet`

preprocessQuantile

The function `preprocessQuantile()` is a useful one-step command to convert directly a basic `RGChannelSet` to a `GenomicRatioSet`, i.e. convert red and green intensities to final beta values, M values, copy number estimates and genomic locations. It also implements a between-array normalization called "Stratified Subset Quantile Normalization". The function has several internal and optional steps available:

1) Fix outliers of both the methylated and unmethylated channels (optional)

2) Remove bad samples using quality control (optional)

3) Map to the genome using `mapToGenome`

4) Perform stratified subset quantile normalization (optional)

5) Predict the sex if not provided using the function `getSex`

We have covered all steps mentioned above except the stratified subset quantile normalization (SSQN). What SSQN does is

- Perform the quantile normalization on the methylated and unmethylated channels separately
- The quantile normalization is applied separately on 3 different subsets of probes: autosomal probes, X-chr probes and Y-chr probes. For the X-chr and Y-chr probes, samples are quantile normalized separately by sex.
- If `stratified = TRUE`, the subset of autosomal probes is further divided into three subsets: probes mapping to CpG islands, probes mapping to CpG shores, and the rest of the probes. Quantile normalization is then applied separately for each subset of probes

```
> gRatioSet.quantile <- preprocessQuantile(RGSet, fixOutliers = TRUE,
+ removeBadSamples = TRUE, badSampleCutoff = 10.5,
+ quantileNormalize = TRUE, stratified = TRUE,
+ mergeManifest = FALSE, sex = NULL)
```

Input: `RGChannelSet`
Output: `GenomicRatioSet`

preprocessFunnorm

The function `preprocessFunnorm()` implements the functional normalization algorithm. Briefly, it uses the internal control probes present on the array to infer the between-array technical variation. It is particularly useful for studies comparing conditions with known large-scale differences, such as

cancer/normal studies, or between tissue studies. It has been shown that for such studies, functional normalization outperforms other existing approaches. By default, is uses the first two principal components of the control probes to infer the unwanted variation. One can change the number of principal components via the option `nPCs`. Suggested use:

```
> gRatioSet.funnorm <- preprocessFunnorm(RGSet)
```

Input: `RGChannelSet`
Output: `GenomicRatioSet`

8 dmpFinder: to find differentially methylated positions (DMPs)

To find differentially methylated positions with respect to a covariate, one can use the function `dmpFinder`. The phenotype may be categorical (e.g. cancer vs. normal) or continuous (e.g. blood pressure). Let's apply `dmrFinder` on the genomicRatioSet.quantile that we have created above, with the predicted sex as the phenotype:

```
> beta <- getBeta(gRatioSet.quantile)
> predictedSex <- pData(gRatioSet.quantile)$predictedSex
> dmp <- dmpFinder(beta, pheno = predictedSex  , type = "categorical")
> head(dmp)
```

9 Bumphunter: to find differentially methylated regions (DMRs)

The `bumphunter` function in `minfi` is a version of the bump hunting algorithm [5] adapted to the 450k array, relying on the `bumphunter` function implemented in the eponym package `bumphunter` [6].

Instead of looking for association between a single genomic location and a phenotype of interest, `bumphunter` looks for association with **genomic regions**. In the context of the 450k array, the algorithm first defines *clusters* of probes. Clusters are simply groups of probes such that two consecutive probe locations in the cluster are not separated by more than some distance

mapGap. To find the differentially methylated regions, the following statistical model is used.

Underlying statistical model for bumphunter

(statistical model described in [6]) We assume that we have data Y_{ij}, here Beta-values of M-values, where i represents biological replicate, and l_j represents a genomic location. The basis statistical model is

$$Y_{ij} = \beta_0(l_j) + \beta_1(l_j)X_j + \epsilon_{ij}$$

with i representing subject, l_j representing the jth location, X_j is the covariate of interest (for example $X_j = 1$ for cases and $X_j = 0$ otherwise), ϵ_{ij} is measurement error, $\beta_0(l)$ is a baseline function, and $\beta_1(l)$ is the parameter of interest, which is a function of location. We assume that $\beta_1(l)$ will be equal to zero over most of the genome, and we want to identify segments where $\beta_1 \neq 0$, which we call **bumps**.

We want to share information between nearby locations, typically through some form of smoothing. As explained in [[[]], the algorithm can be summarized into the following main steps:

A. Compute the coefficients $\beta_1(l_j)$ at each genomic location l_j.

B. Optional smoothing of the coefficients $\beta_1(l_j)$ across genomic locations within a cluster of probes

C. Define a candidate region as a region for which$\beta_1(l_j)$ exceeds a given predefined threshold for all probes within the region

D. Test for significance of the candidate regions using a permutation scheme

Since the permutation test is expensive, parallel computation is implemented in the bumphunter function. The foreach package allows different parallel "back-ends" that will distribute the computation across multiple cores in a single machine, or across machines in a cluster.

```
> pheno <- pData(gRatioSet.quantile)$age
> designMatrix <- model.matrix(~ pheno)
> dmrs <- bumphunter(gRatioSet.quantile, design = designMatrix, cutoff = 0.05, B
> names(dmrs)
> head(dmrs$table, n = 3)
```

10 Sex prediction

By looking at the median total intensity of the X chromosome-mapped probes, denoted $med(X)$, and the median total intensity of the Y-chromosome-mapped probes, denoted $med(Y)$, one can observe two different clusters of points corresponding to which gender the samples belong to. To predict the gender, `minfi` separate the points by using a cutoff on $\log_2 med(Y) - \log_2 med(Y)$. The default cutoff is -2. Since the algorithm needs to map probes to the X-chr and to the Y-chr, the input of the function `getSex()` needs to be a `GenomicMethylSet` or a `GenomicRatioSet`.

```
> predictedSex <- getSex(gRatioSet, cutoff = -2)$predictedSex
> head(predictedSex)
```

To choose the cutoff to separate the two gender clusters, one can plot $med(Y)$ against $med(Y)$ with the function `plotSex`:

```
> plotSex(getSex(gRatioSet, cutoff = -2))
```

Furthermore, `shinyMethyl` has a panel which allow the user to choose interactively the cutoff.

Remark: the function does not handle datasets with only females or only males

References

[1] Hansen, K.D. and Aryee, M. *minfi: Analyze Illumina's 450k methylation arrays. R package version 1.7.8*

[2] *shinyMethyl at shinyMethyl.com*

[3] The Cancer Genome Atlas (TCGA): data portal at https://tcga-data.nci.nih.gov/tcga/

[4] Maksimovic, J., Gordon, L., Oshlack, A. *SWAN: subset quantile Within-Array Normalization for Illumina Infinium HumanMethylation450 BeadChips* Genome Biology, 13(6) R44, 2012

[5] Jaffe, A. E, Murakami, P. Lee, H. Leek, J.T., Fallin, M.D., Feinberg, A.P., Irizarry, R. *Bump hunting to identify differentially methylated regions in epidemiology studies* International Journal of Epidemiology, 41(1): 200-209, 2012

[6] Irizarry, A.R., Aryee, M., Corrada Bravo, H., Hansen, K.D., Jaffee, H.A *bumphunter: Bump Hunter, R package version 1.1.10*

11 Exercises

1) Before processing a RGChannelSet further, coudl you remove the probes which failed more than 50% of the samples in the example dataset?

2) For the top loci that we found differentially methylated for the predicted sex, could you tell if those loci are mostly mapped to the X and Y chromosomes?

3) It is known that the Beta-value distribution of the Type I probes is different from the Beta value distribution of the Type II probes. Can you verify this with by plotting the Beta-value distribution density for each type separately?

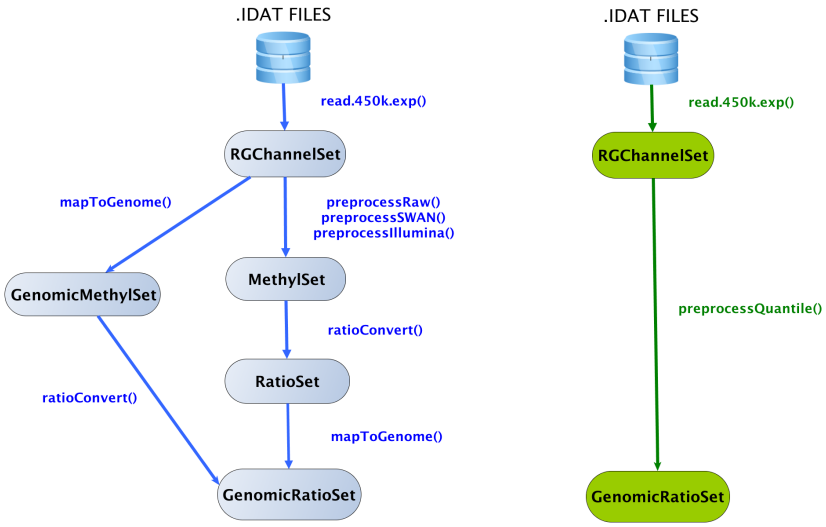

Figure 2: Flow chart of the `minfi`'s functions