DAMEfinder Workflow

true
true

Package version: DAMEfinder

Contents

1 DAMEfinder Overview

1.1 Bumphunting

2 Prerequisites

The differential allele-specific methylation analysis using DAMEfinder starts at the level of the methtuple files. The user must have obtained a list of the files resluting from the methtuple tool to input them in the read_tuples function.

2.1 Illustrative code on how to get from fastq files to the methtuple files level

The DAMEfinder package can be used for detection of differentially allele-specifically methylated regions starting from the files given by the methtuple tool. These files contain unique CpG tuple counts for each sample. We demonstrate how to get to this point starting from fastq files of bisulfite treated reads.

2.1.1 Mapping bisulfite reads

? mention other tools for mapping BSseq reads. Bismark is used to map BS-seq reads to a reference genome. The reads may be trimmed before this step, based on the results of the quality control done on them. If so, then the trimmed reads are mapped using bismark as follows. In the example below, we map the PE reads of sample1 and show other the use of other helpful functions in bismark to exctract methylation information that may be used later in the analysis.

```
# map BS-seq reads
bismark --bowtie2 -p 4 -o aligned_sample1 path_to_reference_genome -1 sample1_R1.fastq.gz -2 sample1_R2.fa

# remove duplicates
```

```
$ deduplicate_bismark  -p aligned_sample1_bismark_bt2_pe.sam




# create sorted bam files
samtools view -Sb aligned_sample1_bismark_bt2_pe.sam >  aligned_sample1_bismark_bt2_pe.bam;
samtools sort aligned_sample1_bismark_bt2_pe.bam aligned_sample1_bismark_bt2_pe_sorted;
samtools index aligned_sample1_bismark_bt2_pe_sorted.bam;







# bismark methylation extractor

bismark_methylation_extractor -p --comprehensive sim_adenoma1_R1.fastq_bismark_bt2_pe.sam;
bismark2bedGraph --counts -o CpG_context_sim_adenoma1_R1.fastq_bismark_bt2_pe.bedGraph CpG_context_sim_ade

# remove duplicates
deduplicate_bismark -p output_sample1.sam

# run methylation_extractor
bismark_methylation_extractor -p --comprehensive NGS-2852_0316_L001_0347_L005_R1_paired.fastq.gz_bismark_b


# create sorted bam files
```

2.1.2 Run Methtuple on bam files

After mapping the BS-seq reads and obtaining the resulting bam files, we can run methtuple. Methtuple requires the input bam files of PE reads to be sorted by query name. For more information on the options in methtuple, refer to the manual of the software. There is the option, for example, of sckipping a number of bases from either end of each read. This is usefult to avoid the first few bases that show methylation bias. The –sc option combines strand information.

```
# Sort bam file by query name
samtools sort -n -@ 10 -m 20G -O bam -T _tmp -o QS_sim_adenoma1_R1.fastq_bismark_bt2_pe_sorted.bam sim_ade

# Run methtuple
methtuple --sc --gzip -m 2 QS_sim_adenoma1_R1.fastq_bismark_bt2_pe_sorted.bam
```

3 Example Workflow

We demonstrate an application of DAMEfinder using the example data set found in the package.

3.1 Read methtuple files

We copy all the files available in the extdata folder of the package which contains the methtuple files we will use in this example. The files are the result of simulated data and encompass a small section of chr14 and contain allele-scpecifically methylated regions in the normal samples, some of which are lost in the adenoma samples.

```
library(DAMEfinder)

file.copy(system.file(package="DAMEfinder", "extdata"), ".", recursive=TRUE)
## [1] TRUE
meta_data <- read.table("extdata/metadata.txt", header = TRUE)
file_list <- list.files(path = "extdata/", pattern = ".tsv.gz")
m <- match(file_list, meta_data$methtuplefile)
file_list <- paste0("extdata/", file_list)
names(file_list) <- meta_data$shortname[m]

file_list
##                                                                adenoma1
## "extdata/adenoma1_QS_sim_R1.fastq_bismark_bt2_pe_sorted.CG.2.tsv.gz"
##                                                                adenoma2
## "extdata/adenoma2_QS_sim_R1.fastq_bismark_bt2_pe_sorted.CG.2.tsv.gz"
##                                                                adenoma3
## "extdata/adenoma3_QS_sim_R1.fastq_bismark_bt2_pe_sorted.CG.2.tsv.gz"
##                                                                 normal1
##  "extdata/normal1_QS_sim_R1.fastq_bismark_bt2_pe_sorted.CG.2.tsv.gz"
##                                                                 normal2
##  "extdata/normal2_QS_sim_R1.fastq_bismark_bt2_pe_sorted.CG.2.tsv.gz"
##                                                                 normal3
##  "extdata/normal3_QS_sim_R1.fastq_bismark_bt2_pe_sorted.CG.2.tsv.gz"
```

```
sample_list <- read_tuples(files = file_list, sample_names = names(file_list))
## Reading extdata/adenoma1_QS_sim_R1.fastq_bismark_bt2_pe_sorted.CG.2.tsv.gz
## Reading extdata/adenoma2_QS_sim_R1.fastq_bismark_bt2_pe_sorted.CG.2.tsv.gz
## Reading extdata/adenoma3_QS_sim_R1.fastq_bismark_bt2_pe_sorted.CG.2.tsv.gz
## Reading extdata/normal1_QS_sim_R1.fastq_bismark_bt2_pe_sorted.CG.2.tsv.gz
## Reading extdata/normal2_QS_sim_R1.fastq_bismark_bt2_pe_sorted.CG.2.tsv.gz
## Reading extdata/normal3_QS_sim_R1.fastq_bismark_bt2_pe_sorted.CG.2.tsv.gz
## Filtering and sorting:
## ......
##  done.

head(sample_list$"adenoma1")
## # A tibble: 6 <U+00D7> 10
##     chr strand     pos1     pos2    MM    MU    UM    UU   cov inter_dist
##   <chr>  <chr>    <int>    <int> <int> <int> <int> <int> <int>      <int>
## 1 chr14      * 93581910 93581916     1     1     2     7    11          6
## 2 chr14      * 93581916 93581925     2     1     2     6    11          9
## 3 chr14      * 93581925 93581933     1     4     5     4    14          8
## 4 chr14      * 93581933 93581936     2     4     2     6    14          3
## 5 chr14      * 93581936 93581938     3     1     3     8    15          2
## 6 chr14      * 93581938 93581940     2     4     1     8    15          2
```

read_tuples returns a list of data frames, with each data frame corresponding to one sample. Each row in the data frame displays a tuple. The chromosome name and starnd are shown followed by pos1 and pos2 which refer to the genomic positions of the first and second CpG in the tuple. The MM, MU, UM, and UU counts of the tuple are displayed where M

stands for methylated and U for unmethylated. For example, UM shows the read counts for the instances where pos1 is unmethylated and pos2 is methylated. The cover and distance between the two genomic positions in the tuple are shown under "cov" and "inter_dist" respectively.

3.1.1 Remove SNPs

On a real data set, tuples that contain CpGs corresponding to SNPs ought to be removed using the remove_snps function. Since the example data set is a simulated one, there is no need to apply the step in our workflow. We demonstrate the use of the function below. A snp_key must be supplied as input, which contains all the SNPs in the following form: 'chr.position.'. For example, "chr5.7382".

suggestiong for future: specifiy the genome only and have the function produce the snp_key

```
sample_list_no_snps <- parallel::mclapply(sample_list, remove_snps, mc.cores=6)
```

3.2 Calculate ASM Score

The calc_asm function takes in the list of samples and outputs a matrix where each row is a tuple and each column is a sample. This matrix contains allele-specific methylation (ASM) scores. The ASM score is a measure of the extent of allele-specific methylation. Equations @ref(eq:asmGeneral), @ref(eq:asmWeight) and @ref(eq:asmTheta) show how the score is calculated. The log odds ratio in equation @ref(eq:asmGeneral) provides a higher score, the more MM and UU counts the tuple has, rather than random methylations of UM and MU. The weight further adds allele-scpecificity where a rather balanced MM:UU count provides the tuple with a higher score.

$$ASM^{(i)} = log\Big\{\frac{X^{(i)}_{MM} \cdot X^{(i)}_{UU}}{X^{(i)}_{MU} \cdot X^{(i)}_{UM}}\Big\} \cdot w_i(\#eq : asmGeneral) \tag{1}$$

$$w_i = P(0.5 - \epsilon < \theta < 0.5 + \epsilon \mid X^{(i)}_{MM}, X^{(i)}_{UU}, \beta_1, \beta_2)(\#eq : asmWeight) \tag{2}$$

$$\theta^{(i)} | X^{(i)}_{MM}, X^{(i)}_{UU}, \beta_1, \beta_2 \sim Beta(\beta_1 + X^{(i)}_{MM}, \beta_2 + X^{(i)}_{UU})(\#eq : asmTheta) \tag{3}$$

where $\theta^{(i)}$ represents the moderated proportion of MM to MM+UU alleles. The weight, w_i is set such that the observed split between MM and UU alleles can depart somewhat from 50/50, while fully methylated or unmethylated tuples, which represents evidence for absence of allele-specificity, are attenuated to 0. The degree of allowed departure can be set according to ϵ, the deviation from 50/50 allowed and the level of moderation, β_1 and β_2. For example, the degree of mixture of subpopulations may play a role (e.g., allele-specificity may not happen on all subpopulations).

```
ASM_score_matrix <- calc_asm(sample_list = sample_list)
## Calculating log odds.
## Calculating ASM score:
## ......
##  done.
## Creating position pair keys:
## ......
##  done.
## Assembling table:
## ......
##  done.
## Transforming.
## Returning SummarizedExperiment with 53514 CpG pairs
```

At this point, the user may want to remove rows (tuples) in the ASM_score_matrix, where all samples of one condition have NA values. One can also set a minimum number of samples to have counts per tuple as a further restriction. However, removing tuples that have NA values in all samples of a particular condition is a necessary step.

comment: add this to the function?

3.3 Get t-Statistics per tuple

Next, we obtain a moderated t-statistic per tuple that reflects a measure of difference between the two conditions being comapred. The t-Statistic is smoothed using smoother function from bumphunter. The midpoint of the two positions in each tuple is set as the genomic position of every tuple in the smoothing process.

```
# Make a design matrix that specifies the two condition per sample
cols <- rep(1,ncol(ASM_score_matrix))
n <- grep("normal", colnames(ASM_score_matrix))
cols[n] <- 0

mod <- matrix(data=c(rep(1,ncol(ASM_score_matrix)), cols), ncol = 2)

# Get t-Statistics
tStatistics <- get_tstats(ASM_score_matrix, mod, method = "ls")
## Calculating moderated t-statistics.
## Warning: Partial NA coefficients for 34481 probe(s)
## Warning: Zero sample variances detected, have been offset away from zero
## Smoothing moderated t-statistics.
## Loading required package: S4Vectors
## Loading required package: stats4
## Loading required package: BiocGenerics
##
## Attaching package: 'BiocGenerics'
## The following objects are masked from 'package:parallel':
##
##     clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
##     clusterExport, clusterMap, parApply, parCapply, parLapply,
##     parLapplyLB, parRapply, parSapply, parSapplyLB
## The following objects are masked from 'package:stats':
##
##     IQR, mad, xtabs
## The following objects are masked from 'package:base':
##
##     Filter, Find, Map, Position, Reduce, anyDuplicated, append,
##     as.data.frame, cbind, colnames, do.call, duplicated, eval,
##     evalq, get, grep, grepl, intersect, is.unsorted, lapply,
##     lengths, mapply, match, mget, order, paste, pmax, pmax.int,
##     pmin, pmin.int, rank, rbind, rownames, sapply, setdiff, sort,
##     table, tapply, union, unique, unsplit, which, which.max,
##     which.min
##
## Attaching package: 'S4Vectors'
## The following objects are masked from 'package:base':
##
##     colMeans, colSums, expand.grid, rowMeans, rowSums
## Loading required package: IRanges
## Loading required package: GenomeInfoDb
```

```
## Loading required package: GenomicRanges
## Loading required package: foreach
## Loading required package: iterators
## Loading required package: locfit
## locfit 1.5-9.1    2013-03-22
## Loading required package: rngtools
## Loading required package: pkgmaker
## Loading required package: registry
##
## Attaching package: 'pkgmaker'
## The following object is masked from 'package:S4Vectors':
##
##     new2
## The following object is masked from 'package:base':
##
##     isNamespaceLoaded
```

3.4 Find DAMEs

Finally, we detect regions that show differential allele-specific methylation. The regionFinder function from bumphunter is used in this process. cutoff explain how it's set

```
dames <- find_dames(tStatistics)
## Warning in bumphunter::regionFinder(x = sm_tstat, chr =
## as.character(GenomeInfoDb::seqnames(sa)), : NAs found and removed. ind
## changed.
## getSegments: segmenting
## getSegments: splitting
## 659 DAMEs found.

head(dames)
##       chr     start       end     value      area cluster indexStart
## 532 chr14 100704945 100706904 -2.239004 371.67462    6493      33743
## 611 chr14 102227139 102228565 -2.079879 193.42873    7865      46531
## 571 chr14 101291043 101294211 -2.165435 175.40021    7050      39600
## 325 chr14  93581757  93582334 -2.064717 158.98318       1         54
## 548 chr14 101004408 101005578 -1.915859 137.94184    6796      36250
## 362 chr14  95235250  95236903 -1.805915  93.90758    1483       7645
##     indexEnd   L clusterL
## 532    34010 268      345
## 611    46698 168      225
## 571    39755 156      201
## 325      146  93      177
## 548    36359 110      173
## 362     7788 144      356
```