DNA Modification Patterns Filtering and Analysis Using DNAModAnnot

Alexis Hardy, Sandra Duharcourt, Matthieu Defrance

Springer Protocols - Methods in Molecular Biology - DNAModAnnot

Alexis Hardy[1], Sandra Duharcourt[2], Matthieu Defrance[1]

[1]Université Libre de Bruxelles, Interuniversity Institute of Bioinformatics in Brussels (IB2), Brussels B-1050, Belgium
[2]Université de Paris, CNRS, Institut Jacques Monod, F-75013 Paris, France

To whom correspondence should be addressed. matthieu.defrance@ulb.be

i. Chapter Title

DNA modification patterns filtering and analysis using DNAModAnnot

ii. Summary/Abstract

Mapping DNA modifications at the base resolution is now possible at the genome level thanks to advances in sequencing technologies. Long-read sequencing data can be used to identify modified base patterns. However, the downstream analysis of Pacific Biosciences (PacBio) or Oxford Nanopore Technologies (ONT) data requires the integration of genomic annotation and comprehensive filtering to prevent the accumulation of artifact signals. We present in this chapter, a linear workflow to fully analyze modified base patterns using the DNA Modification Annotation (DNAModAnnot) package. This workflow includes a thorough filtering based on sequencing quality and false discovery rate estimation and provides tools for a global analysis of DNA modifications. Here, we provide an example of applying this workflow to PacBio data and guide the user by explaining expected outputs via a fully integrated R-markdown script. This protocol is presented with tips showing how to

adapt the provided code for annotating epigenomes of any organism according to the user needs.

iii. Key Words

Epigenomics, Epigenome Annotation, DNA modifications, DNA Methylation, PacBio Sequencing, Nanopore technology, DNAModAnnot.

1. Introduction

Recent advances in sequencing technologies have greatly contributed to the epigenomics field. SMRT sequencing from Pacific Biosciences (PacBio) and nanopore sequencing from Oxford Nanopore Technologies (ONT) allow the mapping of different DNA modifications at the base resolution in the whole genome [1]. These long-read sequencing technologies can even detect modifications in regions with high amount of repeats that were previously inaccessible with Illumina sequencing [1].

For nanopore sequencing software such as Nanopolish or DeepSignal, use differences in electric current intensity to detect modified bases as they pass through the pores [2]. However, for many of these software, the genome annotation (from the GC density to the presence of repeated regions) is essential as it can impact the efficiency of DNA modification detection in ONT data thus leading to higher false positive rates in some genomic regions [2].

For SMRT sequencing, the SMRT-Link software (SMRT-Portal for older versions) uses slowing-down events of the DNA polymerase during sequencing to detect modified bases [1][3]. Several modifications, such as 6-methyladenine (6mA) or 5-methylcytosine (5mC), can be detected as long as the coverage requirement is fulfilled [4]. PacBio also suggests to define a threshold based on the score parameter (also called « Modification QV) by comparing the score of all bases sequenced [3]. However, this method requires a strong signal

that can be easily distinguished from the noise in order to choose an adapted threshold based on the score. Also, SMRT sequencing was found to overestimate the modification levels, especially when the amount of modified bases is very low in the genome [5]. Thus, ill-adapted filtering in such cases can cause high amounts of artifacts.

To overcome this lack of stringency, we have previously released DNAModAnnot (DNA Modification Annotation) [6], a R package allowing comprehensive filtering and analysis of modified patterns for PacBio or ONT data using adapted visualization tools.

This package is divided into 6 modules, as illustrated in Fig. 1, that can be combined to fully analyze pre-processed PacBio or ONT data. DNAModAnnot provides tools to load pre-processed data (« Data Loading ») and analyze the modification distribution at the genome level (« Global DNA Mod ») or using the genome annotation provided (« DNA Mod annotation »). Furthermore « Sequencing quality » assessment and False Discovery Rate estimation (« FDR estimation ») can be directly used to perform a thorough filtering of PacBio or ONT data (« Filter ») (Fig. 1). This modular toolbox uses object classes from the GenomicRanges [7] and BioStrings [8] packages allowing a user-friendly coupling with functions from other main Bioconductor packages.

In this chapter, we provide a roadmap for a systematic analysis of DNA modifications and an example of using DNAModAnnot on PacBio data. This workflow includes the loading of pre-processed files, the filtering steps based on sequencing quality and False Discovery Rate estimations, and the DNA modification pattern analysis with genomic annotations or analysis at the genome-wide level. It provides a summary of the functions provided by DNAModAnnot and a linear processing that can be easily extended with additional R packages for more advanced analyses.

2. Materials

2.1. **Data sources**

In this protocol, we use data from PacBio RSII data [13] (and additional sequencing data listed in Table 1 [13] [14]) to analyze 6mA patterns in *Tetrahymena thermophila*. The patterns of this DNA modification have already been described in this organism [13]. Using this linear workflow, the user will learn to use DNAModAnnot by retrieving and analyzing these patterns.

For SMRT sequencing data, DNAModAnnot need the *modifications.gff* (data from modified bases only) and *modifications.csv* (data from all sequenced bases) files. These pre-processed data can be sourced from https://github.com/AlexisHardy/DNAModAnnot_AdditionalData, also listed in Table 1. Pre-processed files were produced using the SMRT-link-tools v7.0.1.66975-0 and 6-methyladenine (6mA) was detected via the ipdSummary tool [9]. Command-lines to re-generate the *modifications.csv* and *modifications.gff* files from the raw SMRT sequencing data (listed in Table 1) are detailed in the Notes section (*see* **Note** 1).

DNAModAnnot [6] can also load Nanopore (ONT) data pre-processed with the DeepSignal software [10] but we will only focus on PacBio data in this protocol (*see* **Note** 2).

This package also needs the genome assembly (*fasta*) and its annotation (e.g. *gff*) in order to analyze the DNA modification patterns (listed in Table 1).

All the files required to perform the analysis are listed in Table 1. This table also contains additional sequencing data which can be analyzed together with DNA modification patterns.

2.2. Software and installation

The required packages are detailed in the description file of the DNAModAnnot [6] package and can be installed via the install command of the BiocManager [11] package :

```
BiocManager::install(c('Biostrings', 'BSgenome', 'Gviz',
'seqLogo'))
```

DNAModAnnot [6] can be installed via GitHub using the devtools [12] package :

```
devtools::install_github("AlexisHardy/DNAModAnnot")
```

3. Methods

For SMRT sequencing data, methylation detection via SMRT Link [9] returns 2 files: the *modifications.gff* (containing data from modified bases only) and *modifications.csv* (containing data from all sequenced bases) files (*see* **Note** 2 for ONT data) (*see* **Note** 3).

For both sequencing data type, the data must first be loaded into R. Sequencing quality can be assessed to filter out contigs with low coverage which could bias the global statistics of modified bases distribution. DNA modifications distribution can also be analyzed at the genome level. For PacBio data, False Discovery Rate can be estimated to select an appropriate filter based on available detection parameters (score or ipdRatio). By providing the genomic annotation, it is also possible to identify the patterns of DNA modifications associated to specific annotation features.

Here, we provide an example of analysis of 6mA patterns using *Tetrahymena thermophila* PacBio RSII data [13] (and additional sequencing data listed in Table 1 [13] [14]): from the input files importation to the generation of graphs and reports. The results are presented in Fig. 2, Fig 3, and Table 2. A Rmarkdown document is also provided with all commands and details of this protocol (*see* **Note** 4).

3.1. Loading mandatory files

3.1.1. Import Genome sequence information

Go to the webpages listed in Table 1 and collect the *bed*/*csv*/*gff*/*fasta*/*txt* files. For PacBio data, you can either download SMRT link [9] processed files called *modifications.gff* and *modifications.csv* or download the *bax.h5* files via the SRA Run Selector in the GSM2534782 repository and use SMRT Link [9] to generate the *modifications.gff* and *modifications.csv* files (*see* **Note** 1).

1. Import the genome sequence as a *DNAstringSet* object using the `readDNAStringSet` function from the Biostrings package [8] then filter it using the *contig_file.txt* file to keep only the sampled contigs (*see* **Note** 3).

```
organism_genome <-
Biostrings::readDNAStringSet("./genome.fasta")
names(organism_genome) <- gsub(x = names(organism_genome),
                               pattern = " .*",
                               replacement = "")
#We only retrieve the 50 contigs and then we filter the
organism_genome object
contigsToKeep <- read.table("./contig_list.txt")[,1]
organism_genome <- organism_genome[
    names(organism_genome) %in% contigsToKeep]
```

   a. Use the `GetAssemblyReport` function with the genome *DNAstringSet* object to compute a sequencing report about the genome assembly provided (similar to reports from the QUAST software [15]) in a *data.frame* object.

```
report_assembly <- GetAssemblyReport(
  dnastringsetGenome = organism_genome,
  cOrgAssemblyName = "T.thermophila_June2014 (sampled)"
)
```

   b. Use the `GetContigCumulLength` function to retrieve a *data.frame* with the contigs size and cumulated size. This *data.frame* can be used with the `DrawContigCumulLength` function to plot cumulated length by contig.

```
contig_cumul_length <- GetContigCumulLength(organism_genome)
```

```
DrawContigCumulLength(
  nContigCumsumLength = contig_cumul_length$cumsum_Mbp_length,
  cOrgAssemblyName = "T.thermophila_June2014 (sampled)",
  lGridInBackground = TRUE
)
```

2. Use the `GetGenomeGRanges` function with the genome *DNAstringSet* object to retrieve a *GRanges* object representing the contigs (which will be required for some functions from this package).

```
organism_genome_range <- GetGenomeGRanges(organism_genome)
```

3.1.2. Import Modifications input files

1. Use the `ImportPacBioCSV` function to import the *modifications.csv* file as an *Unstitched GPos* object (ModCSV *GPos*) (*see* **Note** 3) (*see* **Note** 5).

```
ModCSV_gpos <- ImportPacBioCSV(
  cPacBioCSVPath = "./modifications.csv",
  cSelectColumnsToExtract = c(
    "refName", "tpl", "strand",
    "base", "score",
    "ipdRatio", "coverage"
  ),
  lKeepExtraColumnsInGPos = TRUE,
  lSortGPos = TRUE,
  cContigToBeAnalyzed = names(organism_genome)
)
```

2. Use the `ImportPacBioGFF` function to import the *modifications.gff* file as a *GRanges* object (ModGFF *GRanges*). Only one modification can be imported from this file which should be defined with the `cNameModToExtract` argument (*see* **Note** 3).

```
ModGFF_granges <- ImportPacBioGFF(
  cPacBioGFFPath = "./modifications.gff",
  cNameModToExtract = "m6A",
  cModNameInOutput = "6mA",
  cContigToBeAnalyzed = names(organism_genome)
)
```

3.2. Sequencing quality assessment and filtering

1. To retrieve the percentage of sequenced bases per contig and per strand, use the `GetSeqPctByContig` function with the ModCSV *GPos* object. This function will return a *list* with the percentage per contig and per strand according to the genome assembly sequence provided.

```
contig_pct_seq <- GetSeqPctByContig(ModCSV_gpos,
  grangesGenome = organism_genome_range
)
```

   a. To plot this percentage using a per contig and per strand barplot, use the `DrawBarplotBothStrands` function and provide the 2 sub-lists (generated on the previous step) corresponding to the forward and the reverse strands (*see* **Note** 6).

```
DrawBarplotBothStrands(
  nParamByContigForward = contig_pct_seq$f_strand$seqPct,
  nParamByContigReverse = contig_pct_seq$r_strand$seqPct,
  cContigNames = contig_pct_seq$f_strand$refName,
  cGraphName = "Percentage of sequencing per contig"
)
```

   b. To remove data from contigs that are not sequenced enough (e.g. less than 95% of sequenced bases), use the `FiltPacBio` function with the sequencing percentage *list* returned by the `GetSeqPctByContig` function (*see* **Note** 7).

2. To look at the global distribution of a numeric parameter, use the `DrawDistriHistBox` function to plot a histogram along a boxplot showing the global range of this parameter.

```
DrawDistriHistBox(ModCSV_gpos$coverage,
  cGraphName = "Coverage distribution of all bases sequenced",
  cParamName = "Coverage",
  lTrimOutliers = FALSE
)
```

3. To compute the mean of the coverage (or any available numeric parameter) per contig and per strand, use the `GetMeanParamByContig` function with the ModCSV

GPos and ModGFF *GRanges* objects depending on which parameter you want to filter on. For coverage, we recommend using the ModCSV *GPos* object.

```
contig_mean <- GetMeanParamByContig(
  grangesData = ModCSV_gpos,
  dnastringsetGenome = organism_genome,
  cParamName = "coverage"
)
```

a. Use the `DrawBarplotBothStrands` function to plot the mean of the chosen parameter per contig and per strand into a barplot (*see* **Note** 6).

```
DrawBarplotBothStrands(
  nParamByContigForward = contig_mean$f_strand$mean_coverage,
  nParamByContigReverse = contig_mean$r_strand$mean_coverage,
  cContigNames = contig_mean$f_strand$refName,
  cGraphName = "Mean Coverage per contig"
)
```

b. Use the `FiltPacBio` function with the mean parameter *list* returned by the `GetMeanParamByContig` function (*see* **Note** 7).

3.3. Analysis of global distribution and motif of DNA modification data

1. Use the `GetModReportPacBio` function to obtain a *data.frame* describing the global distribution of the DNA modification (e.g. Modification counts, ratio, motifs associated, mean of parameters provided) (*see* **Note** 2). You will need to provide both the ModCSV *GPos* and ModGFF *GRanges* objects.

```
report_modifications <- GetModReportPacBio(
  grangesGenome = organism_genome_range,
  grangesPacBioGFF = ModGFF_granges,
  gposPacBioCSV = ModCSV_gpos,
  cOrgAssemblyName = "T.thermophila_June2014 (sampled)",
  dnastringsetGenome = organism_genome,
  cBaseLetterForMod = "A",
  cModNameInOutput = "6mA"
)
```

2. To compute the ratio of modified bases per contig and per strand, use the `GetModRatioByContig` function with the ModGFF *GRanges*. You will also need to provide all the bases that can be targeted (modified or not) by extracting them from

the ModCSV *GPos* object (SubsetModCSV *GPos*). For example, for 6mA you will need to keep only the « A » positions using the following command: `ModCSV_gpos[ModCSV_gpos$base == "A"]`.

```
contig_mod_ratio <- GetModRatioByContig (ModGFF_granges,
  ModCSV_gpos[ModCSV_gpos$base == "A"],
  dnastringsetGenome = organism_genome,
  cBaseLetterForMod = "A"
)
```

a. Use the `DrawBarplotBothStrands` function to plot the ratio of modified bases into a barplot per contig and per strand (*see* **Note** 6).

```
contig_mod_ratio$f_strand$Mod_ratio[
is.na(contig_mod_ratio$f_strand$Mod_ratio)] <- 0
contig_mod_ratio$r_strand$Mod_ratio[
is.na(contig_mod_ratio$r_strand$Mod_ratio)] <- 0

DrawBarplotBothStrands(
  nParamByContigForward = contig_mod_ratio$f_strand$Mod_ratio,
  nParamByContigReverse = contig_mod_ratio$r_strand$Mod_ratio,
  cContigNames = contig_mod_ratio$f_strand$refName,
  cGraphName = "Modif/Base ratio per contig (Sequenced sites
only)"
)
```

b. Use the `FiltPacBio` function with the *list* of modification ratios returned by the `GetModRatioByContig` function if you want to remove contigs with a low modification ratio (*see* **Note** 7).

3. To reveal a potential motif or sequence enrichment around the modified bases, you can use the `DrawModLogo` function (Fig. 2A).

a. You must first retrieve all the trimmed sequences around each modified base using the `GetGRangesWindowSeqandParam` function with the ModGFF *GRanges* object. This will export a new *GRanges* object with a new column named « sequence » where the trimmed sequences will be stored (*see* **Note** 8).

```
ModGFF_granges_seq <-
GetGRangesWindowSeqandParam(ModGFF_granges,
  organism_genome_range,
```

```
  dnastringsetGenome = organism_genome,
  nUpstreamBpToAdd = 5,
  nDownstreamBpToAdd = 5
)
```

b. Then, retrieve the trimmed sequences in the « sequence » column and convert them as a *DNAStringSet* object.

```
Seq_ForLogo <- as(ModGFF_granges_seq$sequence, "DNAStringSet")
```

c. Use this *DNAStringSet* object with the `DrawModLogo` function to plot a logo (Fig. 2A). The genomic background can be provided with the `nGenomicBgACGT` option to correct the logo with the proportion of A, C, G and T respectively in the genome. You can also annotate a few positions on this logo using the `nPositionsToAnnotate` and `cAnnotationText` options to indicate respectively the positions to annotate and the text to be written (*see* **Note** 9).

```
backgroundACGT = c(
  (100-report_assembly["gc_pct",])/2,
  report_assembly["gc_pct",]/2,
  report_assembly["gc_pct",]/2,
  (100-report_assembly["gc_pct",])/2
)/100

DrawModLogo(
  dnastringsetSeqAroundMod = Seq_ForLogo,
  nGenomicBgACGT = backgroundACGT, cYunit = "ic_hide_bg",
  nPositionsToAnnotate = c(6), cAnnotationText = c("6mA"),
nTagTextFontSize = 12
)
```

4. Use the `ExtractListModPosByModMotif` function to retrieve a *list* containing the following elements:

   a. the names of the motifs over-represented with the DNA modification

   b. the same motifs with the position of the modification inside these motifs

   c. a *table* containing the percentage of modifications in each motif tested

d. and a listing of ModGFF *GRanges*: one *GRanges* object for each motif over-represented with the modification (ModGFF *GRangesList*).

```
listMotif_ModGFF_grangeslist <- ExtractListModPosByModMotif(
  grangesModPos = ModGFF_granges,
  grangesGenome = organism_genome_range,
  dnastringsetGenome = organism_genome,
  nUpstreamBpToAdd = 0, nDownstreamBpToAdd = 1,
  nModMotifMinProp = 0.05,
  cBaseLetterForMod = "A",
  cModNameInOutput = "6mA"
)
```

The minimum proportion required to define motifs as "over-represented" with the modification can be modified using the `nModMotifMinProp` option (*see* **Note** 8).

5. Extract all the bases that can be targeted (modified or not) from the ModCSV *GPos* object and convert it as a *GRanges* object (SubsetModCSV *GRanges*). For example, for 6mA, you will need to keep only the « A » positions using the following command:

```
SubsetModCSV_granges <- as(ModCSV_gpos[ModCSV_gpos$base ==
"A"], "GRanges")
```

Then, retrieve all the trimmed sequences around each targeted base using the `GetGRangesWindowSeqandParam` function with the SubsetModCSV *GRanges* object (*see* **Note** 8).

```
SubsetModCSV_granges_seq <- GetGRangesWindowSeqandParam(
  grangesData = SubsetModCSV_granges,
  grangesGenome = organism_genome_range,
  dnastringsetGenome = organism_genome,
  nUpstreamBpToAdd = 0,
  nDownstreamBpToAdd = 1
)
```

3.4. False Discovery Rate estimations and filtering (PacBio only)

DNAModAnnot [6] provides tools to estimate False Discovery Rate (FDR) based on a threshold for parameters associated to modification detection [16] (*see* **Note** 10). These FDR

estimations can guide the choice of the filters to be used on the score or ipdRatio parameters. FDR estimation is only available for PacBio data.

1. Use the SubsetModCSV *GRanges* object from the previous part with the `GetFdrEstListByThresh` function to estimate the False Discovery Rate by threshold on a parameter to be filtered (defined with the `cNameParamToTest` option; usually the score or the ipdRatio for PacBio data).

```
score_fdr_by_motif_list <- GetFdrEstListByThresh(
  grangesDataWithSeq = SubsetModCSV_granges_seq,
  grangesDataWithSeqControl = NULL,
  cNameParamToTest = "score",
  nRoundDigits = 1,
  cModMotifsAsForeground =
listMotif_ModGFF_grangeslist$motifs_to_analyse
)
```

   The `GetFdrEstListByThresh` function will return a *list* (by motif over-represented) of *data.frames*. Each *data.frame* contains the FDR estimated by threshold (for the provided parameter) and the adjusted FDR (≈ the cumulative minimum FDR). 2 methods to estimate the FDR are provided with this function [16]:

   a. If you have a control sample (i.e. a non-methylated sample, for example Whole-Genome Amplified/PCR Amplified) you can provide it via the `grangesDataWithSeqControl` option.

      In this case, FDR will be estimated using the data provided via the `grangesDataWithSeqControl` option as the background signal (*see* **Note** 10).

      The data provided via the `grangesDataWithSeqControl` option must have the same format as the SubsetModCSV *GRanges* object with the sequence of the modified sample. This means that, after methylation detection, the control sample (initially as a *modifications.csv* file) must be imported using

the `ImportPacBioCSV` function as a control ModCSV *GPos*. Steps and filters applied to the sample ModCSV should also be applied to the control data to ensure a correct FDR estimation.

b. If you do not have a control sample, you must then leave the `grangesDataWithSeqControl` option empty.

In this case, you can estimate the FDR by comparing motifs associated to modifications against other motifs to be used as the background signal (*see* **Note** 10).

Here, motifs associated to the DNA modification (over-represented motifs) must be provided via the `cModMotifsAsForeground` argument as a *character vector*. FDR will be estimated for each over-represented motif separately. The user can choose to look at one motif in particular. It is also possible to test all « over-represented » motifs by retrieving the "motifs_to_analyse" *vector* from the output *list* of the `ExtractListModPosByModMotif` function.

This function returns a *list* with one *data.frame* if a control sample is provided. If not a *list* with one *data.frame* per motif tested will be returned. Each *data.frame* contains the FDR estimation per threshold on the parameter tested.

2. To retrieve the threshold associated to a user-defined FDR, use the `GetFdrBasedThreshLimit` function with the FDR estimation *list* from the previous step (*see* **Note** 11).

```
score_fdr_by_motif_limit <-
GetFdrBasedThreshLimit(score_fdr_by_motif_list,
  nFdrPropForFilt = 0.05,
  lUseBestThrIfNoFdrThr = TRUE
)
```

3. Use the `DrawFdrEstList` function with the FDR estimation *list* to plot FDR estimation distribution per threshold and per motif, along with a user-defined FDR limit/value to be represented on the graph.

```
DrawFdrEstList(
  listFdrEstByThr = score_fdr_by_motif_list,
  cNameParamToTest = "score",
  nFdrPropForFilt = 0.05
)
```

4. Use the `FiltPacBio` function with the output of the `GetFdrBasedThreshLimit` function to filter the ModGFF object according to the defined FDR-associated threshold (by motif or not). In this case, the ModGFF *GRangesList* (returned by the `ExtractListModPosByModMotif` function) must be used for filtering (especially if no control sample was provided during FDR estimation) (*see* **Note** 7). Only two parameters are recognized for FDR estimation and filtering here the ipdRatio and the score.

```
ModGFF_grangeslist <- FiltPacBio(
  grangesPacBioGFF =
listMotif_ModGFF_grangeslist$GRangesbyMotif,
  listFdrEstByThrIpdRatio = NULL,
  listFdrEstByThrScore = score_fdr_by_motif_limit
)$gff
```

3.5. Analysis of DNA modification patterns with genomic annotations and other sequencing data

In this section, genomic annotations must be provided to analyze the modified base distribution. Modified base counts or proportions can be computed for any category of genomic features or quantitative parameters and can be compared to other sequencing data, such as MNase-seq data.

In this part, we define Mod as the modified bases and Base as all target bases (modified or not) that use the same motifs as Mod. For example, for Mod defined as “6mAT” (“6mA” in AT motif), Base would be “AT” (“A” in AT motif).

In this part, 3 objects will be required for most tools (along with the genome sequence imported as a DNAStringSet object):

1. A ModGFF *GRanges* associated to a motif:
    a. (PacBio only) Extract the *GRanges* for the motif to analyze from the *GRangesList* provided by the `FiltPacBio` function used on ModGFF *GRanges* after False Discovery Rate estimation.

```
ModGFF_granges <- ModGFF_grangeslist[["AT"]]
```

    b. Or extract the *GRanges* for the motif to analyze from the *GRangesList* within the *list* provided by the `ExtractListModPosByModMotif` function used on ModGFF *GRanges*.

```
ModGFF_granges <-
listMotif_ModGFF_grangeslist$ModGFF_grangeslist[["AT"]]
```

    c. Or use the `GetGRangesWindowSeqandParam` function with the ModGFF *GRanges* then subset on the column « sequence » using the motif to analyze (*see* **Note** 8).

```
ModGFF_granges <-
ModGFF_granges_seq[ModGFF_granges_seq$sequence == "AT",]
```

2. A ModCSV *GRanges* associated to a motif.

    Retrieve the ModCSV *GRanges* object with the sequence (returned by the `GetGRangesWindowSeqandParam` function) then subset on the column « sequence » using the motif to analyze.

```
ModCSV_granges <-
ModCSV_granges_seq[ModCSV_granges_seq$sequence == "AT",]
```

3. A *GRanges* object filled with the genome annotation to be compared to.

    a. For annotation files using *gff* format, use the `readGFFAsGRanges` function from the rtracklayer package [17] to import the annotation into a *GRanges* object.

```
annotations_path <- "./T_thermophila_June2014.gff3"
annot_range <- rtracklayer::readGFFAsGRanges(annotations_path)
```

    b. Then, use the `PredictMissingAnnotation` function to add « intergenic » features to the new *GRanges* object. For some functions, the feature "intergenic" will be required for comparison between genes and intergenic regions. If your annotation file also provides mRNA positions and exon (or intron) positions, the `PredictMissingAnnotation` function can add the missing annotation (introns or exons) to the new *GRanges* object.

```
annot_range <- PredictMissingAnnotation(
  grangesAnnotations = annot_range,
  grangesGenome = organism_genome_range,
  cFeaturesColName = "type",
  cGeneCategories = c("gene"),
  lAddIntronRangesUsingExon = TRUE
)
```

3.5.1. Computing counts by genomic feature

1. Use the `GetModBaseCountsByFeature` function (with the annotation *GRanges* and the ModGFF/ModCSV *GRanges* associated with a motif) to count Base and Mod for each feature provided in the annotation *GRanges* (annotation *GRanges* with ModBase counts).

```
annot_range_MBcounts <- GetModBaseCountsByFeature(
  grangesAnnotations = annot_range,
  grangesModPos = ModGFF_granges,
  gposModTargetBasePos = SubsetModCSV_granges,
  lIgnoreStrand = FALSE
)
```

2. Use the `DrawModBasePropByFeature` function (with the annotation *GRanges* and the ModGFF/ModCSV *GRanges* associated with a motif) to compare the proportion of Base and Mod between different annotation categories (Fig. 2B). Features to be compared must be listed as a *character vector* in the `cFeaturesToCompare` option.

```
DrawModBasePropByFeature(
  grangesAnnotationsWithCounts = annot_range_MBcounts,
  cFeaturesToCompare = c("gene", "intergenic"),
  lUseCountsPerkbp = TRUE,
  cBaseMotif = "AT",
  cModMotif = "6mAT"
)
```

3.5.2. Quantitative parameter by feature and by Mod counts categories

It is also possible to compare a quantitative parameter with Base and Mod counts in genomic features.

1. Retrieve the annotation *GRanges* with ModBase counts returned by the `GetModBaseCountsByFeature` function.
2. Import or compute the parameter that you want to compare to the Mod or Base counts. For example, we imported RNA-seq file containing read counts per gene in this protocol using the `read.table` function.

```
expression_file_path <- "./GSM692081_Growth.map.txt"
expression_dataframe <- read.table(
  file = expression_file_path,
  header = TRUE, sep = "\t"
)
```

3. The quantitative parameter to be compared with the Mod or Base must be loaded as a new column within the annotation *GRanges* with ModBase counts. For example, we filtered the annotation *GRanges* with ModBase counts to keep only the genes then we used the `merge` function with the `mcols()` of the annotation *GRanges* with

ModBase counts to replace its `mcols()`. In this example, we also normalize the counts of mapped RNA-seq reads using the size of the genes.

```
genes_range_MBcounts_param <-
annot_range_MBcounts[annot_range_MBcounts$type == "gene"]
genes_range_MBcounts_param <- genes_range_MBcounts_param[
  genes_range_MBcounts_param$Name %in%
expression_dataframe$Gene_ID
]
GenomicRanges::mcols(genes_range_MBcounts_param) <- merge(
  x = GenomicRanges::mcols(genes_range_MBcounts_param),
  by.x = "Name",
  y = expression_dataframe,
  by.y = "Gene_ID"
)
genes_range_MBcounts_param$Number_of_mapped_reads_perkbp <-
  1000*genes_range_MBcounts_param$Number_of_mapped_reads /
  GenomicRanges::width(genes_range_MBcounts_param)
```

4. Use the `DrawParamPerModBaseCategories` function to plot the distribution of the quantitative parameter provided by category of Mod and Base counts (Fig. 2D).

```
DrawParamPerModBaseCategories(
  grangesAnnotationsWithCounts = genes_range_MBcounts_param,
  cParamColname = "Number_of_mapped_reads",
  cParamFullName = "Gene expression at G-m (mid-log
exponential growth)",
  cParamYLabel = "RNA-seq read counts (G-m)",
  cSelectFeature = "gene",
  lUseCountsPerkbp = FALSE,
  cBaseMotif = "AT",
  cModMotif = "6mAT",
  lBoxPropToCount = FALSE, lUseSameYAxis = TRUE
)
```

3.5.3. Computing counts within genomic features

1. Use the `GetModBaseCountsWithinFeature` function (with the annotation *GRanges* and the ModGFF/ModCSV *GRanges* associated with a motif) to count Mod and Base within segments of each genomic feature provided. Each feature provided is cut into a specific number of windows (defined by the `nWindowsNb` argument) and counts are returned for each window of each feature. Here, we filtered the annotation

GRanges to keep only the genes before using the `GetModBaseCountsWithinFeature` function.

```
genes_range <- annot_range[annot_range$type == "gene", ]
genes_range <- GetModBaseCountsWithinFeature(
  grangesAnnotations = genes_range,
  grangesModPos = ModGFF_granges,
  gposModTargetBasePos = SubsetModCSV_granges,
  lIgnoreStrand = FALSE,
  nWindowsNb = 20
)
```

2. Then use the `DrawModBaseCountsWithinFeature` function to represent the distribution within provided features through a barplot (Fig. 2C).

```
DrawModBaseCountsWithinFeature(
  grangesAnnotationsWithCountsByWindow = genes_range,
  cFeatureName = "gene",
  cBaseMotif = "AT",
  cModMotif = "6mAT"
)
```

3.5.4. Computing distance from genomic features

1. Use the `GetDistFromFeaturePos` function with the annotation *GRanges* and the ModGFF *GRanges* to retrieve, for each feature provided, the distance, in bp, between this feature and a Mod (using a window of specific size around each feature) (*see* **Note** 12).

```
Mod_distance_feature_countslist <- GetDistFromFeaturePos(
  grangesAnnotations = annot_range,
  cSelectFeature = "gene",
  grangesData = ModGFF_granges,
  lGetGRangesInsteadOfListCounts = FALSE,
  lGetPropInsteadOfCounts = TRUE,
  cWhichStrandVsFeaturePos = "both",
  nWindowSizeAroundFeaturePos = 600,
  lAddCorrectedDistFrom5pTo3p = TRUE,
  cFeaturePosNames = c("TSS", "TTS")
)
```

2. Repeat the previous step with the annotation *GRanges* and the ModCSV *GRanges* instead to retrieve, for each feature provided, the distance, in bp, between this feature and a Base (*see* **Note** 12).

```
Base_distance_feature_countslist <- GetDistFromFeaturePos(
  grangesAnnotations = annot_range,
  cSelectFeature = "gene",
  grangesData = SubsetModCSV_granges,
  lGetGRangesInsteadOfListCounts = FALSE,
  lGetPropInsteadOfCounts = TRUE,
  cWhichStrandVsFeaturePos = "both",
nWindowSizeAroundFeaturePos = 600,
  lAddCorrectedDistFrom5pTo3p = TRUE,
  cFeaturePosNames = c("TSS", "TTS")
)
```

3. Use the `DrawModBasePropDistFromFeature` function with the output from the 2 previous steps to plot the proportion of Mod or Base around genomic features provided (Fig. 3).

```
DrawModBasePropDistFromFeature(
  listModCountsDistDataframe =
Mod_distance_feature_countslist,
  listBaseCountsDistDataframe =
Base_distance_feature_countslist,
  cFeaturePosNames = c("TSS", "TTS"),
  cBaseMotif = "AT",
  cModMotif = "6mAT"
)
```

a. It is also possible to add at least 1 additional axis on the plot to compare modification signal with another parameter by using the `AddToModBasePropDistFromFeaturePlot` function (Fig. 3). However, distances can only be computed with *GPos* objects or *GRanges* with a size of 1bp per window. For *GRanges* using windows with a size > 1bp, use the `GetGposCenterFromGRanges` function to retrieve the central position of each window (*see* **Note** 13).

```
bedfile_path <- "./GSM2534785_SB210_MNase.120_260.unique.bed"
bedfile_object <- rtracklayer::import.bed(bedfile_path)
bedfile_object <- GetGposCenterFromGRanges(bedfile_object)
```

b. Use the `GetDistFromFeaturePos` function with the annotation *GRanges* and the output from the previous step then the `AddToModBasePropDistFromFeaturePlot` function to plot this

parameter against Mod and Base proportions (Fig. 3) (*see* **Note** 13). A new axis will thus be added on the previous plot from the `DrawModBasePropDistFromFeature` function unless the plot is no longer available (*see* **Note** 14).

```
bedfile_distance_feature_countslist <- GetDistFromFeaturePos(
  grangesAnnotations = annot_range,
  cSelectFeature = "gene",
  grangesData = bedfile_object,
  lGetGRangesInsteadOfListCounts = FALSE,
  lGetPropInsteadOfCounts = FALSE,
  cWhichStrandVsFeaturePos = "both",
nWindowSizeAroundFeaturePos = 600,
  lAddCorrectedDistFrom5pTo3p = TRUE,
  cFeaturePosNames = c("TSS", "TTS")
)
AddToModBasePropDistFromFeaturePlot(
  dPosCountsDistFeatureStart =
bedfile_distance_feature_countslist[[1]],
  dPosCountsDistFeatureEnd =
bedfile_distance_feature_countslist[[2]],
  cSubtitleContent = "Along with MNase-seq read center
distance",
  cParamYLabel = "MNase-seq read center count",
  cParamColor = "cyan3",
  lAddAxisOnLeftSide = TRUE, cParamLty = 1, cParamLwd = 2
)
```

3.5.5. Local visualization with Gviz

DNAModAnnot [6] provides several functions which can be used alongside the Gviz [18] package for local visualization (*see* Gviz [18] documentation for main functions).

1. The `ExportFilesForGViz` function allows the user to export files which can be used for streaming (except for the *gff3* format) with the `plotTracks` function from Gviz package [18]. Here, using the *bam* format, it is possible to use Gviz [18] streaming options also for genomic annotation.
2. To display the genomic annotations using streaming (using the adapted *bam* file), use the `ImportBamExtendedAnnotationTrack` function as the import function

(via the `stream` and `importFunction` options while making the annotation track). In this case, to allow the names of the genomic features to be displayed, the "mapping" sub-list of the generated annotation track must be manually completed with the new "id" and "group" values defined in the previous step (see Note 4).

4. Notes

1. To generate the *modifications.csv* and *modifications.gff* files using SMRT-Link-tools v7.0.1.66975-0:

    a. download the *fasta* genome and index the genome:

```
samtools faidx "genome.fasta"
```

    b. For each SMRT cell to analyze, download the *bax.h5* files (file names must finish with the "*h5*") then retrieve the *bam* file:

```
bax2bam "SMRT_file_basename".{1,2,3}.bax.h5 -o
"SMRT_file_basename"
```

    Then, align the reads from the *bam* file to the *fasta* genome:

```
pbalign -vvv --nproc "Number of available processors" --
algorithm blasr --forQuiver --byread -metrics
DeletionQV,IPD,InsertionQV,PulseWidth,QualityValue,MergeQV,Sub
stitutionQV,DeletionTag --tmpDir /tmp
"SMRT_file_basename".subreads.bam "genome.fasta"
"SMRT_file_basename"_aligned.bam
```

    c. Merge all SMRTcells:

```
samtools merge SMRT_merged.bam *_aligned.bam
```

    d. Index the merged file:

```
pbindex SMRT_merged.bam
```

    e. Then launch the DNA modification detection tool:

```
ipdSummary --verbose --methylFraction --minCoverage 25 --
mapQvThreshold 30 --identify m6A --numWorkers "Number of
available processors" --outfile tmp_dir/ SMRT_merged_6mA --
reference "genome.fasta" SMRT_merged.bam
```

This command returns the *modifications.csv* and *modifications.gff* files. With this command, they would be called *SMRT_merged_6mA.csv* and *SMRT_merged_6mA.gff* respectively. This command can easily take days if run locally and/or with a low number of available CPUs. If you want to analyze some contigs, use the `--refContigs` option with the name of the contigs that you want to analyze. It is also possible to use a loop to launch methylation detection by group of contigs.

2. To analyze DNA modification patterns, ONT data must be processed with the DeepSignal software [10]. Follow the steps on the DeepSignal GitHub repository for the ONT data processing: https://github.com/bioinfomaticsCSU/deepsignal [19].

   After the full pre-processing of nanopore data via DeepSignal, you must retrieve the modification-frequency file generated via the *call_modification_frequency.py* script.

   Use the `ImportDeepSignalModFrequency` function to load the modification-frequency file as an *Unstitched GPos* object (ModCSV *GPos*).

   Then, use the `FiltDeepSignal` function with the ModCSV *GPos* object to retrieve a ModGFF *GPos* by simply filtering target sites which have a fraction ("frac") above 0.

```
ModGFF_gpos <- FiltDeepSignal(
    gposDeepSignalModBase = ModCSV_gpos,
    cParamNameForFilter = "frac",
    nFiltParamLoBoundaries = 0,
    nFiltParamUpBoundaries = 1,
    cFiltParamBoundariesToInclude = "upperOnly"
)$Mod)
```

   a. The `FiltDeepSignal` function returns a *list* with the filtered ModCSV as the first element of the *list*, and the filtered ModGFF as the second element of the *list*.

b. Some options of this function require ModCSV or the ModGFF *GPos* objects or the sequence of the genome (`dnastringsetGenome` option).

c. You can filter out contigs based on different conditions using a combination of the following arguments (ModCSV *GPos* object is required for most of these options): `cContigToBeRemoved`, `nContigMinSize`, `listPctSeqByContig`, `nContigMinPctOfSeq`, `listMeanCovByContig`, `nContigMinCoverage`, `cParamNameForFilter`, `listMeanParamByContig`, `nContigFiltParamLoBound`, `nContigFiltParamUpBound`.

d. You can filter out modifications from the ModGFF object based on different conditions using a combination of the following arguments: `cParamNameForFilter`, `nFiltParamLoBoundaries`, `nFiltParamUpBoundaries`, `cFiltParamBoundariesToInclude`, `nModMinCoverage`.

The ModCSV *GPos* and the ModGFF *GPos* objects from ONT-DeepSignal data can then be used in place of the ModCSV *GPos* and the ModGFF *GRanges*/*GPos* objects from PacBio data except for the False Discovery Rate estimation functions. For some functions, the ModCSV and ModGFF *GPos* objects will first need to be converted as *GRanges* objects. Also, when using ONT data, the functions `GetModReportPacBio` and `FiltPacBio` must be replaced by the functions `GetModReportDeepSignal` and `FiltDeepSignal` respectively.

3. For this example, only data from 50 random contigs will be analyzed: the listing of selected contigs is available in the *contig_list.txt* file listed in Table 1.

4. A Rmarkdown document is given in the supplementary data and on the GitHub of the DNAModAnnot [6] with all the commands used: https://github.com/AlexisHardy/DNAModAnnot_AdditionalData.
5. If the `SortGPos` argument is TRUE, the importation will take a longer time but the size of the *GPos* object will be highly reduced.
6. By default, the `lIsOrderedLargestToSmallest` argument is TRUE and order the contigs from the largest to the smallest on the x-axis.
7. The `FiltPacBio` function is a wrapper function for filtering PacBio data contained in ModGFF and ModCSV objects. For ONT data, check the `FiltDeepSignal` function (*see* **Note** 2).
    a. This function returns a *list* with the filtered ModCSV as the first element of the *list*, and the filtered ModGFF as the second element of the *list*. If the user provides a ModGFF *GRangesList* instead of *GRanges*, the second element of the *list* will also be returned as a *GRangesList*.

```
Mod_filtered_data <- FiltPacBio(
  gposPacBioCSV = ModCSV_gpos,
  grangesPacBioGFF = ModGFF_granges,
  cContigToBeRemoved = NULL,
  dnastringsetGenome = organism_genome,
  nContigMinSize = 1000,
  listPctSeqByContig = contig_percentage_sequencing,
  nContigMinPctOfSeq = 95,
  listMeanCovByContig = contig_mean_coverage,
  nContigMinCoverage = 10
)
ModCSV_gpos <- Mod_filtered_data$csv
ModGFF_granges <- Mod_filtered_data$gff
```

    b. Providing the ModGFF is mandatory. Some options of this function also require ModCSV *GPos* object or the sequence of the genome (`dnastringsetGenome` option).

c. It is possible to filter out contigs based on different conditions using a combination of the following arguments (ModCSV *GPos* object is required for most of these options): `cContigToBeRemoved`, `nContigMinSize`, `listPctSeqByContig`, `nContigMinPctOfSeq`, `listMeanCovByContig`, `nContigMinCoverage`, `cParamNameForFilter`, `listMeanParamByContig`, `nContigFiltParamLoBound`, `nContigFiltParamUpBound`.

d. It is also possible to filter out modifications from the ModGFF object based on different conditions using a combination of the following arguments: `cParamNameForFilter`, `nFiltParamLoBoundaries`, `nFiltParamUpBoundaries`, `cFiltParamBoundariesToInclude`, `nModMinCoverage`, `nModMinIpdRatio`, `nModMinScore`.

e. Finally, it is possible to filter out modifications from the ModGFF *GRangesList* object based on False Discovery Rate estimations by providing the output *list* of the `GetFdrBasedThreshLimit` function to the `listFdrEstByThrIpdRatio` or `listFdrEstByThrScore` arguments depending on which parameter has been used for the estimation of the False Discovery Rate: ipdRatio or score respectively.

8. Use the `nUpstreamBpToAdd` and `nDownstreamBpToAdd` arguments to choose the size of the sequence motif (that includes the target base) to look at or to filter.
9. The Y-axis can be changed to plot information content or probabilities using the `cYunit` option. You can also remove the depletion signal by deactivating the `lPlotNegYAxis` option. Sequences that do not have full width and sequences that have some N or some gaps "-" are automatically removed before drawing the sequence plot. If a base is enriched 100% at one position, this base alone will be

represented, and other bases will not be represented into the « depletion » part unless if the 'prob' value is used for the y-axis: here the complementary base would then be represented.

10. Formulas can be found in the documentation of the `GetFdrEstListByThresh` function and are based on formulas recently published [16].
11. The chosen FDR is defined by the `nFdrPropForFilt` argument (default to 5%). The threshold will be defined as the closest value below this level of FDR for each motif (or only for some motifs if the `lUseBestThrIfNoFdrThr` argument is used).
12. If the genomic annotation feature provided is larger than 1bp, its two extremities will be used instead for computing the distance. If the `lGetGRangesInsteadOfListCounts` argument is deactivated, the `GetDistFromFeaturePos` function will return instead a *list* of *data.frames* giving the counts (or proportion) of Mod (or Base) by distance toward the feature.
13. For example, we imported MNase-seq data from a *bed* file using the `import.bed` function from the rtracklayer package [17] then we used the `GetGposCenterFromGRanges` function before using its output with the `GetDistFromFeaturePos` function with the output of the `import.bed` function. Then we used the `AddToModBasePropDistFromFeaturePlot` function to plot this parameter against Mod and Base proportions.
14. Up until two additional parameters can be added on this plot, use the `lAddAxisOnLeftSide` argument to choose on which side to put the axis of the new parameter to add.

5. References

1. Gouil Q, Keniry A (2019) Latest techniques to study DNA methylation. Essays in Biochemistry 63:639–648 . https://doi.org/10.1042/EBC20190027

2. Liu Y, Rosikiewicz W, Pan Z, Jillette N, Wang P, Taghbalout A, Foox J, Mason C, Carroll M, Cheng A, Li S (2021) DNA methylation-calling tools for Oxford Nanopore sequencing: a survey and human epigenome-wide evaluation. Genome Biology 22:295 . https://doi.org/10.1186/s13059-021-02510-z

3. Methylome Analysis Technical Note · PacificBiosciences/Bioinformatics-Training Wiki. In: GitHub. https://github.com/PacificBiosciences/Bioinformatics-Training. Accessed 28 Jan 2022

4. Detecting DNA base modifications using single molecule, real-time sequencing. White Pap Base Modif. 2015. https://www.pacb.com/wp-content/uploads/2015/09/WP_Detecting_DNA_Base_Modifications_Using_SMRT_Sequencing.pdf. Accessed 27 Jan 2022

5. O'Brown ZK, Boulias K, Wang J, Wang SY, O'Brown NM, Hao Z, Shibuya H, Fady P-E, Shi Y, He C, Megason SG, Liu T, Greer EL (2019) Sources of artifact in measurements of 6mA and 4mC abundance in eukaryotic genomic DNA. BMC Genomics 20:445 . https://doi.org/10.1186/s12864-019-5754-6

6. Hardy A, Matelot M, Touzeau A, Klopp C, Lopez-Roques C, Duharcourt S, Defrance M (2021) DNAModAnnot: a R toolbox for DNA modification filtering and annotation. Bioinformatics 37:2738–2740 . https://doi.org/10.1093/bioinformatics/btab032

7. Lawrence M, Huber W, Pagès H, Aboyoun P, Carlson M, Gentleman R, Morgan MT,

Carey VJ (2013) Software for Computing and Annotating Genomic Ranges. PLOS Computational Biology 9:e1003118 . https://doi.org/10.1371/journal.pcbi.1003118

8. Pagès H, Aboyoun P, Gentleman R, DebRoy S (2022) Biostrings: Efficient manipulation of biological strings. https://bioconductor.org/packages/Biostrings/. Accessed 26 Jan 2022

9. PacBio - Software Downloads. In: PacBio. https://www.pacb.com/support/software-downloads/. Accessed 26 Jan 2022

10. Ni P, Huang N, Zhang Z, Wang D-P, Liang F, Miao Y, Xiao C-L, Luo F, Wang J (2019) DeepSignal: detecting DNA methylation state from Nanopore sequencing reads using deep-learning. Bioinformatics 35:4586–4595 . https://doi.org/10.1093/bioinformatics/btz276

11. Morgan M, Ramos M (2021) BiocManager: Access the Bioconductor Project Package Repository. https://CRAN.R-project.org/package=BiocManager. Accessed 26 Jan 2022

12. Wickham H, Hester J, Chang W, Bryan J (2021) devtools: Tools to Make Developing R Packages Easier. https://CRAN.R-project.org/package=devtools. Accessed 26 Jan 2022

13. Wang Y, Chen X, Sheng Y, Liu Y, Gao S (2017) N6-adenine DNA methylation is associated with the linker DNA of H2A.Z-containing well-positioned nucleosomes in Pol II-transcribed genes in Tetrahymena. Nucleic Acids Research 45:11594–11606 . https://doi.org/10.1093/nar/gkx883

14. Xiong J, Lu X, Zhou Z, Chang Y, Yuan D, Tian M, Zhou Z, Wang L, Fu C, Orias E, Miao W (2012) Transcriptome Analysis of the Model Protozoan, Tetrahymena thermophila, Using Deep RNA Sequencing. PLOS ONE 7:e30630 . https://doi.org/10.1371/journal.pone.0030630

15. Gurevich A, Saveliev V, Vyahhi N, Tesler G (2013) QUAST: quality assessment tool for genome assemblies. Bioinformatics 29:1072–1075 . https://doi.org/10.1093/bioinformatics/btt086

16. Zhu S, Beaulaurier J, Deikus G, Wu TP, Strahl M, Hao Z, Luo G, Gregory JA, Chess A, He C, Xiao A, Sebra R, Schadt EE, Fang G (2018) Mapping and characterizing N6-methyladenine in eukaryotic genomes using single-molecule real-time sequencing. Genome Res 28:1067–1078 . https://doi.org/10.1101/gr.231068.117

17. Lawrence M, Gentleman R, Carey V (2009) rtracklayer: an R package for interfacing with genome browsers. Bioinformatics 25:1841–1842 . https://doi.org/10.1093/bioinformatics/btp328

18. Hahne F, Ivanek R (2016) Visualizing Genomic Data Using Gviz and Bioconductor. In: Mathé E, Davis S (eds) Statistical Genomics: Methods and Protocols. Springer, New York, NY, pp 335–351

19. Ni P, Huang N, bioinformaticsCSU (2021) DeepSignal. https://github.com/bioinfomaticsCSU/deepsignal. Accessed 26 Jan 2022

Tables

Table 1
List of input files used in this protocol

Description	Format	Link	Required?
T. thermophila (June2014) genome assembly sequence	*fasta*	http://ciliate.org/index.php/home/downloads	Mandatory
T. thermophila (June2014) genome annotation	*gff3*	http://ciliate.org/index.php/home/downloads	Mandatory
T. thermophila pre-processed SMRT-seq data (via SMRT-link tools v7.0.1.66975-0) using T_thermophila_June2014 genome assembly. SMRT-seq data was retrieved from GSM2534782 [13] contig_list.txt contains the listing of contigs selected in this example.	*gff*, *csv* and *txt*	https://github.com/AlexisHardy/DNAModAnnot_AdditionalData	Mandatory
T. thermophila SMRT-seq data (to be retrieved via SRA Run Selector) [13]	*bax.h5*	https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSM2534782	Not required if pre-processed files are available
T. thermophila MNase-seq [13]	*bed*	https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSM2534785	Optional
T. thermophila H2A.Z ChIP-seq [13]	*bed*	https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSM2534783	Optional
T. thermophila RNA-seq [14]	*txt*	https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSM692081	Optional

Table 2
Global 6mA distribution report using a subset of *T. thermophila* SMRT-seq data and T.thermophila_June2014 genome assembly (small version)

Parameters	T.thermophila_June2014
6mA count	19,354
Adenine count (sequenced)	15,488,227
Ratio 6mA/A	0.00125
Ratio 6mA/A corrected	0.00096
6mA mean fraction	0.76926
6mA mean coverage	31.38
6mA mean ipdRatio	43.26
6mA mean identificationQv	24.26
6mAA %	0.67%
6mAC %	0.19%
6mAG %	0.72%
6mAT %	98.42%

Figures

Fig. 1
Overall workflow of the DNAModAnnot package used to filter and analyze DNA modification patterns

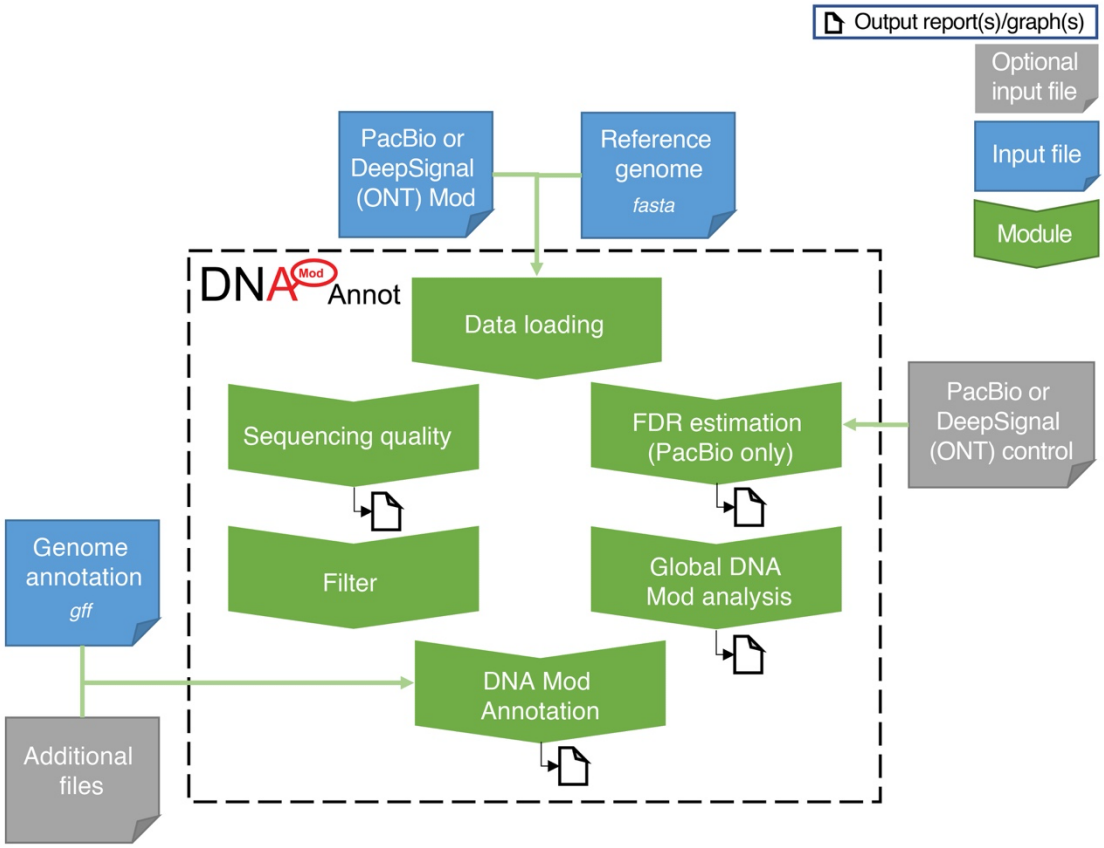

Fig. 2

Example of graphs generated via DNAModAnnot using a subset of *T. thermophila* SMRT-seq data. (A) Logo generated via the DrawModLogo function displaying an AT motif associated with 6mA. 5bp were selected upstream and downstream 6mA positions. "Negative" information content corresponds to information content for depletion signal. (B) Barplot generated using the DrawModBasePropByFeature function using gene and intergenic categories for comparison showing 6mAT enrichment in genes. (C) Barplot generated using the DrawModBaseCountsWithinFeature function showing the enrichment of 6mAT downstream TSS. (D) Boxplot generated using the DrawParamPerModBaseCategories function: a slight association between 6mAT count (per kbp) and normalized RNA-seq read count can be observed. G-m (mid-log exponential growth) sample was used here. Intervals are computed from quantiles to optimize the repartition of windows between the categories.

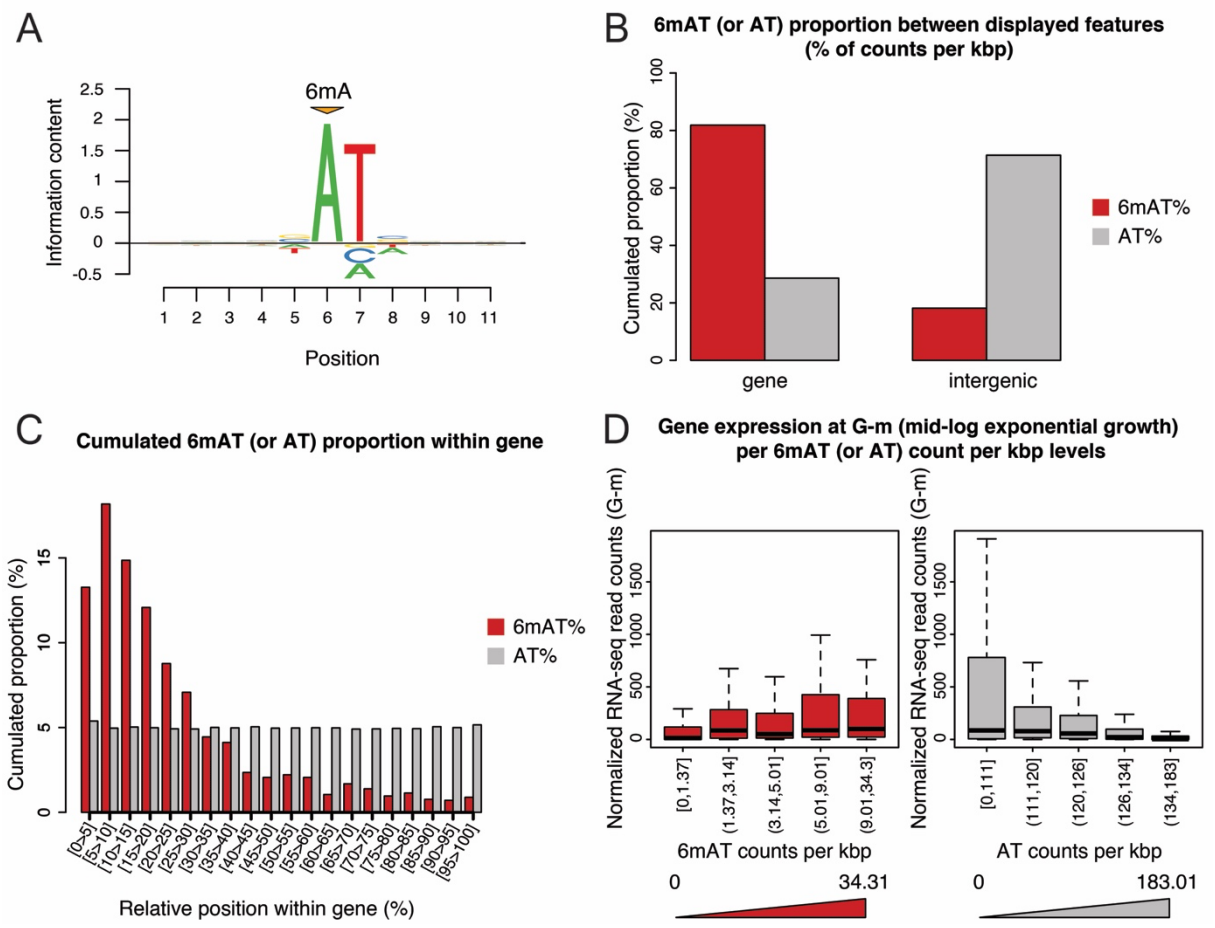

Fig. 3
Example of graph generated using the DrawModBasePropDistFromFeature function with a subset of *T. thermophila* SMRT-seq data then the AddToModBasePropDistFromFeaturePlot function with *T. thermophila* MNase-seq data. Here, enrichment of 6mAT can be observed between peaks of MNase-seq reads downstream TSS.