m6anet

Feb 24, 2023

Contents

m6anet is a python tool that leverages Multiple Instance Learning framework to detect m6a modifications from Nanopore Direct RNA Sequencing data.

m6anet requires Python version 3.7 or higher. To install the latest release with PyPI (recommended) run:

```
pip install m6anet
```

See our *Installation page* for details.

To detect m6A modifications from your direct RNA sequencing sample, you can follow the instructions in our *Quick-start page*. m6Anet is trained on dataset sequenced using the SQK-RNA002 kit and has been validated on dataset from SQK-RNA001 kit. Newer pore version might alter the raw squiggle and affect segmentation and classification results and in such cases m6Anet might need to be retrained.

CHAPTER 1

Contents

1.1 Installation

m6Anet requires Python version 3.7 or higher to run. Installation typically takes less than 5 minutes but might vary depending on your connection speed

1.1.1 PyPI installation (recommended)

```
pip install m6anet
```

1.1.2 Installation from our GitHub repository

```
git clone https://github.com/GoekeLab/m6anet.git
cd m6anet
python setup.py install
```

1.2 Quick Start

1.2.1 Dataprep

m6Anet dataprep requires eventalign.txt from `nanopolish eventalign`:

```
nanopolish eventalign --reads reads.fastq --bam reads.sorted.bam --genome transcript.
↪fa --scale-events --signal-index --summary /path/to/summary.txt  --threads 50 > /
↪path/to/eventalign.txt
```

This function segments raw fast5 signals to each position within the transcriptome, allowing m6Anet to predict modification based on the segmented signals. In order to run eventalign, users will need: * `reads.fastq`: fastq file generated from basecalling the raw .fast5 files * `reads.sorted.bam`: sorted bam file obtained from aligning reads.fastq to the reference transcriptome file * `transcript.fa`: reference transcriptome file

We have also provided a demo eventalign.txt dataset in the repository under /path/to/m6anet/m6anet/tests/data/eventalign.txt. Please see Nanopolish for more information.

After running nanopolish eventalign, we need to preprocess the segmented raw signal file using 'm6anet dataprep':

```
m6anet dataprep --eventalign /path/to/m6anet/m6anet/tests/data/eventalign.txt \
                --out_dir /path/to/output --n_processes 4
```

The output files are stored in `/path/to/output`:

- `data.json`: json file containing the features to feed into m6Anet model for prediction
- `data.log`: Log file containing all the transcripts that have been successfully preprocessed
- `data.info`: File containing indexing information of data.json for faster file access and the number of reads for each DRACH positions in eventalign.txt
- `eventalign.index`: Index file created during dataprep to allow faster access of Nanopolish eventalign.txt during dataprep

1.2.2 Inference

Once `m6anet dataprep` finishes running, we can run `m6anet inference` on the dataprep output

```
m6anet inference --input_dir path/to/output --out_dir path/to/output  --n_processes 4␣
↪--num_iterations 1000
```

m6anet inference will run default human model trained on the HCT116 cell line. In order to run Arabidopsis-based model, change the inputs to `--model_state_dict`, `--read_proba_threshold` and `norm_path` to

```
m6anet inference --input_dir path/to/output --out_dir path/to/output  --read_proba_
↪threshold 0.0032978046219796 \
  --model_state_dict m6anet/m6anet/model/model_states/arabidopsis_virc.pt --norm_
↪path m6anet/m6anet/model/norm_factors/norm_factors_virc.joblib\
  --n_processes 4 --num_iterations 1000
```

m6Anet will sample 20 reads from each candidate site and average the probability of modification across several round of sampling according to the –num_iterations parameter. The output file *data.site_proba.csv* contains the probability of modification at each individual position for each transcript. The output file will have 6 columns

- `transcript_id`: The transcript id of the predicted position
- `transcript_position`: The transcript position of the predicted position
- `n_reads`: The number of reads for that particular position
- `probability_modified`: The probability that a given site is modified
- `kmer`: The 5-mer motif of a given site
- `mod_ratio`: The estimated percentage of reads in a given site that is modified

The output file *data.indiv_proba.csv* contains the probability of modification for each read

- `transcript_id`: The transcript id of the predicted position
- `transcript_position`: The transcript position of the predicted position

- `read_index`: The read identifier from nanopolish that corresponds to the actual read_id from nanopolish summary.txt
- `probability_modified`: The probability that a given read is modified

The total run time should not exceed 10 minutes on a normal laptop. We also recommend a threshold of 0.9 for selecting m6A sites based on the `probability_modified` column, which can be relaxed at the expense of having lower model precision.

m6Anet also supports pooling over multiple replicates. To do this, simply input multiple folders containing m6anet-dataprep outputs:

```
m6anet inference --input_dir data_folder_1 data_folder_2 ... --out_dir output_folder -
↪-n_processes 4 --num_iterations 1000
```

1.3 Command line arguments

We provide 2 main scripts to run m6A prediction as the following.

1.3.1 `m6anet dataprep`

- Input

Output files from `nanopolish eventalign`. Please refer to *Quickstart page* for more details on running nanopolish.

Argument name	Required	Default value	Description
–eventalign=FILE	Yes	NA	Eventalign filepath, the output from nanopolish.
–out_dir=DIR	Yes	NA	Output directory.
–n_processes=NUM	No	1	Number of processes to run.
–chunk_size=NUM	No	1000000	chunksize argument for pandas read csv function on the eventalign input
–readcount_max=NUM	No	1000	Maximum read counts per gene.
–readcount_min=NUM	No	1	Minimum read counts per gene.
–skip_index	No	False	To skip indexing the eventalign nanopolish output, can only be used if the index has been created before
–n_neighbors=NUM	No	1	The number of flanking positions to process
–min_segment_count=NUM	No	1	Minimum read counts over each candidate m6A segment
–compress	No	False	Round down output features to 3 decimal places

- Output

File name	File type	Description
eventalign.index	csv	File index indicating the position in the *eventalign.txt* file (the output of nanopolish eventalign) where the segmentation information of each read index is stored, allowing a random access.
data.json	json	Intensity level mean for each position.
data.index	csv	File index indicating the position in the *data.json* file where the intensity level means across positions of each gene is stored, allowing a random access.
data.readcount	csv	Summary of readcounts per gene.

1.3.2 `m6anet inference`

- Input

Output files from `m6anet dataprep`.

Argument name	Required	Default value	Description
–input_dir=DIR	Yes	NA	Input directory that contains data.json, data.index, and data.readcount from m6anet-dataprep
–out_dir=DIR	Yes	NA	Output directory for the inference results from m6anet
–model_config=FILE	No	prod_pooling.toml	Model architecture specifications. Please see examples in m6anet/model/configs/model_configs/prod_pooling.toml
–model_state_dict=FILE	No	prod_pooling_pr_auc.pt	Model weights to be used for inference. Please see examples in m6anet/model/model_states/
–batch_size=NUM	No	64	Number of sites to be loaded each time for inference
–n_processes=NUM	No	1	Number of processes to run.
–num_iterations=NUM	No	5	Number of times m6anet iterates through each potential m6a sites.
–read_proba_threshold=NUM	No	0.033379376	Threshold for each individual read to be considered modified during stoichiometry calculation

- Output

File name	File type	Description
data.site_proba.csv	csv	Result table for each candidate m6A site
data.indiv_proba.csv	csv	Result table for each candidate m6A read

1.3.3 `m6anet train`

Argument name	Required	Default value	Description
–model_config=FILE	Yes	NA	Model architecture specifications. Please see examples in m6anet/model/configs/model_configs/prod_pooling.toml
–train_config=FILE	Yes	NA	Config file for training the model. Please see examples in m6anet/model/configs/training_configs/oversampled.toml
–save_dir=DIR	Yes	NA	Save directory to save the training results
–device=STR	No	cpu	Device to use for training the model. Set to cuda:cuda_id if using GPU
–lr=NUM	No	4e-4	Learning rate for the ADAM optimizer
–seed=NUM	No	25	Random seed for model training
–epochs=NUM	No	50	Number of epochs to train the model.
–num_workers=NUM	No	1	Number of processes to run.
–save_per_epoch=NUM	No	10	Number of recurring epoch to save the model
–weight_decay=NUM	No	0	Weight decay parameteter for the ADAM optimizer
–num_iterations=NUM	No	5	Number of times m6anet iterates through each potential m6a sites.

1.4 Training m6Anet

m6Anet expects a training config file and a model config file, both on TOML format. We have provided examples of the model config file and the training config file in:

- m6anet/m6anet/model/configs/model_configs/m6anet.toml
- m6anet/m6anet/model/configs/training_configs/m6anet_train_config.toml

Below is the content of m6anet_train_config.toml

```
[loss_function]
loss_function_type = "binary_cross_entropy_loss"

[dataset]
root_dir = "/path/to/m6anet-dataprep/output"
min_reads = 20
norm_path = "/path/to/m6anet/m6anet/model/norm_factors/norm_dict.joblib"
num_neighboring_features = 1

[dataloader]
    [dataloader.train]
    batch_size = 256
    sampler = "ImbalanceOverSampler"

    [dataloader.val]
    batch_size = 256
    shuffle = false

    [dataloader.test]
    batch_size = 256
    shuffle = false
```

User can modify some basic training information such as the batch_size, the number of neighboring features, as well as the minimum number of reads per site to train m6Anet. We have also calculated the normalization factors required under norm_path variable. In principle, one can even change the loss_function_type by choosing one from m6anet/m6anet/utils/loss_functions.py or defining a new one. Sampler can be set to ImbalanceOverSampler (in which the model will perform oversampling to tackle the data imbalance with m6Anet modification) or any other sampler from m6anet/m6anet/utils/data_utils.py

The training script will look for data.info.labelled file and data.json file under the root_dir directory. While data.info can be obtained by running m6anet dataprep on nanopolish eventalign.txt file, data.info.labelled must be supplied by the user by adding extra columns to the data.info file produced by m6anet dataprep. Additionally, data.info.labelled must be of the following format:

```
transcript_id   transcript_position n_reads start end  modification_status set_type
ENST00000361055 549                 11      0     940  0                   Train
ENST00000361055 554                 12      940   1969 0                   Train
ENST00000475035 133                  3      1969  2294 0                   Train
ENST00000222329 309                 11      2299  3284 0                   Val
ENST00000222329 2496                15      3284  4593 0                   Val
ENST00000222329 2631                23      4593  6548 0                   Val
ENST00000523944 72                   1      6548  6665 0                   Test
ENST00000523944 2196                14      6665  7853 0                   Test
```

Here modification status tells the model which positions are modified and which positions are not modified. The column set_type informs the training script which part of the data we should train on and which part of the data should be used for validation and testing purpose. Lastly, n_reads corresponds to the number of reads that comes from the corresponding transcript positions and any sites with n_reads less than the min_reads specified in he training config file will not be used for training validation, or testing. We have also provided an example of data.readcount.labelled in m6anet/demo/ folder.

Below is the content of m6anet.toml:

```
model = "prod_sigmoid_pooling"

[[block]]
block_type = "DeaggregateNanopolish"
num_neighboring_features = 1

[[block]]
block_type = "KmerMultipleEmbedding"
input_channel = 66
output_channel = 2
num_neighboring_features = 1

[[block]]
block_type = "ConcatenateFeatures"

[[block]]
block_type = "Linear"
input_channel = 15
output_channel = 150
activation = "relu"
batch_norm = true

[[block]]
block_type = "Linear"
input_channel = 150
output_channel = 32
activation = "relu"
batch_norm = false

[[block]]
block_type = "SigmoidProdPooling"
input_channel = 32
n_reads_per_site = 20
```

The training script will build the model block by block. For additional information on the block type, please check the source code under m6anet/m6anet/model/model_blocks

In order to train m6Anet, please change the root_dir variable inside prod_pooling.toml to m6anet/demo/. Afterwards, run m6anet-dataprep:

```
m6anet dataprep --eventalign m6anet/demo/eventalign.txt \
               --out_dir m6anet/demo/ --n_processes 4
```

This will produce data.index file and data.json file that will be used for the script to access the preprocessed data Next, to train m6Anet using the demo data, run:

```
m6anet train --model_config m6anet/model/configs/model_configs/m6anet.toml --train_
↪config ../m6anet/model/configs/training_configs/m6anet_train_config.toml --save_dir␣
↪/path/to/save_dir --device cpu --lr 0.0001 --seed 25 --epochs 30 --num_workers 4 --
↪save_per_epoch 1 --num_iterations 5
```

The model will be trained on cpu for 30 epochs and we will save the model states every 1 epoch. One can replace the device argument with cuda to train with GPU. For complete description of the command line arguments, please see *Command line arguments page*

1.5 Getting Help

We appreciate your feedback and questions! You can report any error or suggestion related to m6Anet as an issue on github. If you have questions related to the manuscript, data, or any general comment or suggestion please use the Discussions.

Thank you!

1.6 Release Note 2.0.0

1.6.1 API Changes

The m6Anet functions for preprocessing, inference, and training have now been simplified. We now provide a single entry point for all m6anet functionalities through the m6anet module. This means that all the old functionalities of m6Anet are now available through the m6anet module call, such as `m6anet dataprep`, `m6anet inference` and `m6anet train` functions. The command `m6anet-dataprep`, `m6anet-run_inference` and `m6anet-train` are deprecated and will be removed in the next version. Please check our updated *Quickstart page* and *Training page* for more details on running m6Anet.

We have also made some changes to the m6anet dataprep function. Previously m6anet-dataprep produces data.index and data.readcount files to run inference, and we realized that this can be simplified by combining the two files together. The current m6anet dataprep (and also the deprecated m6anet-dataprep) now produces a single data.info file that combines the information from both data.index and data.readcount. Furthermore, m6anet inference (also the deprecated m6anet-run_inference) now requires data.info file to be present in the input directory. We have also provided a function for users to convert older dataprep output files to the newest format using

```
m6anet convert --input_dir /path/to/old/dataprep/output --out_dir /path/to/old/
↪dataprep/output
```

This function will create data.info file by combining the old data.index and data.readcount files. The users still need to make sure that data.info file is located in the same folder as data.json file

1.6.2 Faster and Better Inference Implementation

In order to minimize the effect of sequencing depth in m6Anet prediction, a fixed number of reads are sampled from each site during m6Anet training. This process is repeated during inference where the sampling will be repeated several times for each candidate site to stabilize the modification probability. The number of sampling rounds is controlled through the option *–num_iterations* and the default was set to 5 in the previous version of m6Anet to minimize running time.

A low number of sampling iterations results in unstable probability value for individual sites and while the overall performance of m6Anet on large datasets remains unaffected, users looking to identify and study modifications on individual sites will benefit from a more consistent modification score. In m6Anet 2.0.0, we have improved the inference process so that it can accommodate a higher number of sampling iterations while still maintaining a relatively fast inference time. Here we include the comparison between the older m6Anet version against the current release in terms of their peak memory usage and running time over a different number of sampling rounds on our HEK293T dataset with 95030 sites and 8019824 reads. The calculation is done on AMD EPYC 7R32 with *–num_processes* set to 25.

Version Number	Peak Memory Usage(MB)	Running Time(s)	Number of Iterations
m6Anet v-1.1.1	480.5	8876.77	50
m6Anet v-1.1.1	677.9	18009.92	100
m6Anet v-2.0.0	553.7	392.91	5
m6Anet v-2.0.0	571.3	229.92	50
m6Anet v-2.0.0	576.4	409.71	100
m6Anet v-2.0.0	578.5	408.17	1000

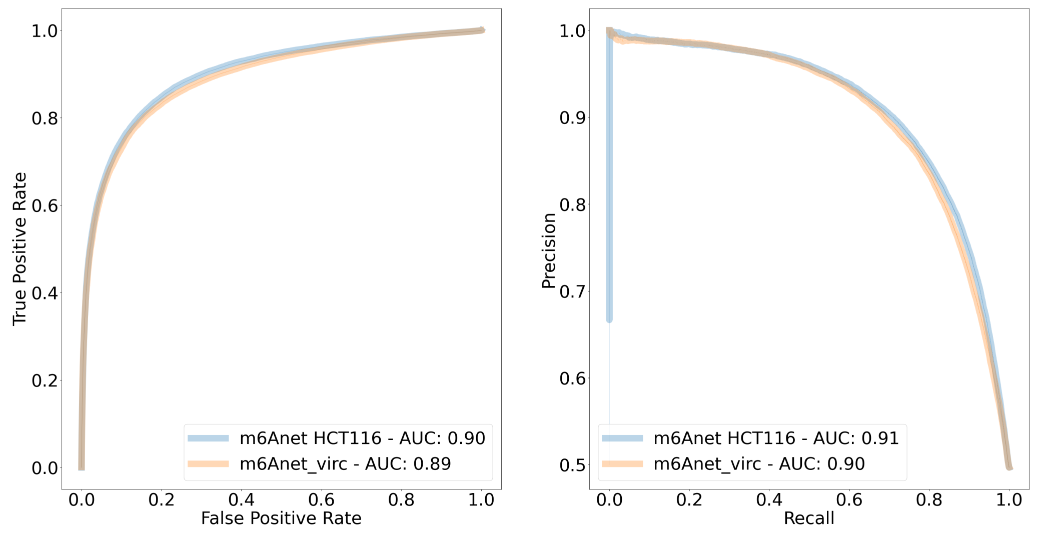
As we can see, the latest version of m6Anet has relatively constant peak memory usage with minimal difference in running time between 100 and 1000 iteration runs. This is done by saving each individual read probability in *data.indiv_proba.csv* file before sampling the required amount of reads for each site in parallel.

1.6.3 Rounding of Dataprep Output

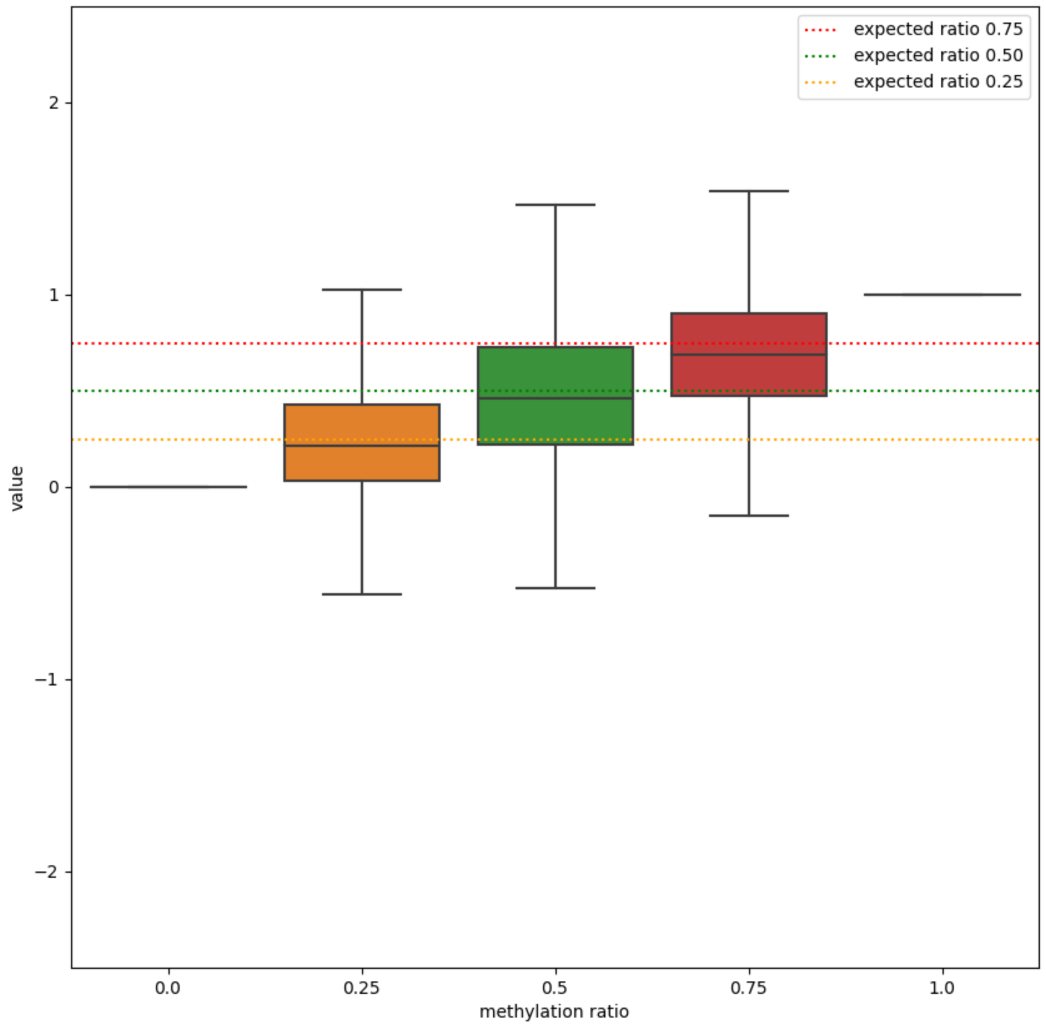
Users can now add `--compress` flag to `m6anet dataprep` to round the dataprep output features to 3 decimal places. In our experience, this reduces the file size for data.json significantly without compromising model performance.

1.6.4 Arabidopsis Trained m6Anet

We have also included m6Anet model trained on the Arabidopsis VIRc dataset from our paper as an option for users who are looking to study m6A modification on plant genomes or to aggregate predictions from different m6Anet models on their datasets. Here we present single molecular probability results on synthetic RNA from the curlcake dataset



The single-molecule m6A predictions of the Arabidopsis model seem to be comparable with the human model with ROC AUC of 0.89 and PR AUC of 0.90 on the synthetic. We also validate the ability to predict per-molecule modifications of the Arabidopsis model on the human HEK293T METTL3-KO and wild-type samples that were mixed to achieve an expected relative m6A stoichiometry of 0%, 25%, 50%, 75%, and 100% from xPore on the sites predicted to be modified in wild-type samples (probability ≥ 0.7)) As we can see, from the 1041 shared sites that we inspect across the HEK293T mixtures, the median prediction of the model follows the expected modification ratio.



In order to run the Arabidopsis model, please add the following command when running m6anet inference

- `--read_proba_threshold : 0.0032978046219796`
- `--model_state_dict : m6anet/m6anet/model/model_states/arabidopsis_virc.pt`
- `--norm_path : m6anet/m6anet/model/norm_factors/norm_factors_virc.joblib`

CHAPTER 2

Citing m6Anet

If you use m6Anet in your research, please cite [Christopher Hendra, et al.,Detection of m6A from direct RNA sequencing using a Multiple Instance Learning framework. *Nat Methods* (2022)](https://doi.org/10.1038/s41592-022-01666-1)

CHAPTER 3

Contacts

m6anet is developed and maintained by Christopher Hendra and Jonathan Göke from the Genome Institute of Singapore, A*STAR. If you want to contribute, please leave an issue in our repo

Thank you!