1 Contents
  1.1 Installation ......................................................... 3
  1.2 Quick Start ....................................................... 3
  1.3 Command line arguments ........................................... 4
  1.4 Training m6Anet .................................................. 6
  1.5 Getting Help ....................................................... 8
2 Contacts ................................................................. 9
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 3.8 or higher. To install the latest release with PyPI (recommended) run:

```
ip 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.
1.1 Installation

m6Anet requires Python version 3.8 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

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

We have also provided a demo dataset in the repository under /path/to/m6anet/demo/eventalign.txt.

Firstly, we need to preprocess the segmented raw signal file in the form of nanopolish eventalign file using `m6anet-dataprep`:

```
m6anet-dataprep --eventalign m6anet/demo/eventalign.txt \
--out_dir /path/to/output --n_processes 4
```
The output files are stored in /path/to/output:

- data.index: Indexing of data.json to allow faster access to the file
- 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.readcount: File containing 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

Now we can run m6anet over our data using m6anet-run_inference:

```
m6anet-run_inference --input_dir demo_data --out_dir demo_data --infer_mod-rate --n_processes 4
```

The output files demo_data/data.result.csv.gz contains the probability of modification at each individual position for each transcript. The output file will have 4 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

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

```
m6anet-run_inference --input_dir demo_data_1 demo_data_2 ... --out_dir demo_data --infer_mod-rate --n_processes 4
```

### 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
### m6anet

#### 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
--read_count_max=NUM | No | 1000 | Maximum read counts per gene.
--read_count_min=NUM | No | 1 | Minimum read counts per gene.
--index | No | True | 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

#### Output

<table>
<thead>
<tr>
<th>File name</th>
<th>File type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eventalign.index</td>
<td>csv</td>
<td>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.</td>
</tr>
<tr>
<td>data.json</td>
<td>json</td>
<td>Intensity level mean for each position.</td>
</tr>
<tr>
<td>data.index</td>
<td>csv</td>
<td>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.</td>
</tr>
<tr>
<td>data.readcount</td>
<td>csv</td>
<td>Summary of readcounts per gene.</td>
</tr>
</tbody>
</table>

### 1.3.2 m6anet-run_inference

#### Input

Output files from m6anet-dataprep.

<table>
<thead>
<tr>
<th>Argument name</th>
<th>Required</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--input_dir=DIR</td>
<td>Yes</td>
<td>NA</td>
<td>Input directory that contains data.json, data.index, and data.readcount from m6anet-dataprep</td>
</tr>
<tr>
<td>--out_dir=DIR</td>
<td>Yes</td>
<td>NA</td>
<td>Output directory for the inference results from m6anet</td>
</tr>
<tr>
<td>--model_config=FILE</td>
<td>No</td>
<td>prod_pooling.toml</td>
<td>Model architecture specifications. Please see examples in m6anet/model/configs/model_configs/prod_pooling.toml</td>
</tr>
<tr>
<td>--model_state_dict=FILE</td>
<td>No</td>
<td>prod_pooling_pr_auc.pt</td>
<td>Model weights to be used for inference. Please see examples in m6anet/model/model_states/</td>
</tr>
<tr>
<td>--batch_size=NUM</td>
<td>No</td>
<td>64</td>
<td>Number of sites to be loaded each time for inference</td>
</tr>
<tr>
<td>--n_processes=NUM</td>
<td>No</td>
<td>1</td>
<td>Number of processes to run.</td>
</tr>
<tr>
<td>--num_iterations=NUM</td>
<td>No</td>
<td>5</td>
<td>Number of times m6anet iterates through each potential m6a sites.</td>
</tr>
<tr>
<td>--infer_mod_rate</td>
<td>No</td>
<td>False</td>
<td>Whether to output m6A modification stoichiometry for each candidate site</td>
</tr>
<tr>
<td>--read_proba_threshold</td>
<td>No</td>
<td>0.033379376</td>
<td>Threshold for each individual read to be considered modified during stoichiometry calculation</td>
</tr>
</tbody>
</table>

#### Output
### 1.3.3 m6anet-train

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

### 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 training config and model config file in:

- m6anet/m6anet/model/configs/model_configs/prod_pooling.toml
- m6anet/m6anet/model/configs/training_configs/oversampled.toml

Below is the content of oversampled.toml

```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]
```
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.readcount.labelled` file and `data.index` file under the `root_dir` directory. While `data.index` can be obtained by running `m6anet-dataprep` over `nanopolish eventalign.txt` file, `data.readcount.labelled` must be supplied by the user by adding extra columns to the `data.readcount` file produced by `m6anet-dataprep`. Additionally, `data.readcount.labelled` must be of the following format:

<table>
<thead>
<tr>
<th>transcript_id</th>
<th>transcript_position</th>
<th>n_reads</th>
<th>modification_status</th>
<th>set_type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENST00000361055</td>
<td>549</td>
<td>11</td>
<td>0</td>
<td>Train</td>
</tr>
<tr>
<td>ENST00000361055</td>
<td>554</td>
<td>12</td>
<td>0</td>
<td>Train</td>
</tr>
<tr>
<td>ENST00000475035</td>
<td>133</td>
<td>3</td>
<td>0</td>
<td>Train</td>
</tr>
<tr>
<td>ENST00000222329</td>
<td>309</td>
<td>11</td>
<td>0</td>
<td>Val</td>
</tr>
<tr>
<td>ENST00000222329</td>
<td>2496</td>
<td>15</td>
<td>0</td>
<td>Val</td>
</tr>
<tr>
<td>ENST00000222329</td>
<td>2631</td>
<td>23</td>
<td>0</td>
<td>Val</td>
</tr>
<tr>
<td>ENST00000523944</td>
<td>72</td>
<td>1</td>
<td>0</td>
<td>Test</td>
</tr>
<tr>
<td>ENST00000523944</td>
<td>2196</td>
<td>14</td>
<td>0</td>
<td>Test</td>
</tr>
</tbody>
</table>

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 the 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 `prod_pooling.toml`:

```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
```

(continues on next page)
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/prod_pooling.toml -- \
  --train_config ../m6anet/model/configs/training_configs/oversampled.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!
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!