Nanopolish Documentation

Release 0.8.4

Simpson Lab

Oct 21, 2019

Contents

nanopolish is a software package for signal-level analysis of Oxford Nanopore sequencing data. Nanopolish can calculate an improved consensus sequence for a draft genome assembly, detect base modifications, call SNPs and indels with respect to a reference genome and more (see Nanopolish modules, below).

CHAPTER 1

Installation

1.1 Dependencies

A compiler that supports C++11 is needed to build nanopolish. Development of the code is performed using gcc-4.8.

By default, nanopolish will download and compile all of its required dependencies. Some users however may want to use system-wide versions of the libraries. To turn off the automatic installation of dependencies set *HDF5=noinstall*, *EIGEN=noinstall* or *HTS=noinstall* parameters when running *make* as appropriate. The current versions and compile options for the dependencies are:

- libhdf5-1.8.14 compiled with multi-threading support `--enable-threadsafe`
- eigen-3.2.5
- htslib-1.4

Additionally the helper *scripts* require biopython and pysam.

1.2 Installing the latest code from github (recommended)

You can download and compile the latest code from github as follows

```
git clone --recursive https://github.com/jts/nanopolish.git
cd nanopolish
make
```

1.3 Installing a particular release

When major features have been added or bugs fixed, we will tag and release a new version of nanopolish. If you wish to use a particular version, you can checkout the tagged version before compiling

```
git clone --recursive https://github.com/jts/nanopolish.git
cd nanopolish
git checkout v0.7.1
make
```

1.4 To run using docker

First build the image from the dockerfile:

```
docker build .
```

Note the uuid given upon successful build. Then you can run nanopolish from the image:

```
docker run -v /path/to/local/data/data/:/data/ -it :image_id  ./nanopolish eventalign␣
↪-r /data/reads.fa -b /data/alignments.sorted.bam -g /data/ref.fa
```

CHAPTER 2

Quickstart - how to polish a genome assembly

The original purpose of nanopolish was to improve the consensus accuracy of an assembly of Oxford Nanopore Technology sequencing reads. Here we provide a step-by-step tutorial to help you get started.

Requirements:

- nanopolish
- samtools
- minimap2
- MUMmer

2.1 Download example dataset

You can download the example dataset we will use here:

```
wget http://s3.climb.ac.uk/nanopolish_tutorial/ecoli_2kb_region.tar.gz
tar -xvf ecoli_2kb_region.tar.gz
cd ecoli_2kb_region
```

Details:

- Sample : E. coli str. K-12 substr. MG1655
- Instrument : MinION sequencing R9.4 chemistry
- Basecaller : Albacore v2.0.1
- Region: "tig00000001:200000-202000"
- Note: Ligation-mediated PCR amplification performed

This is a subset of reads that aligned to a 2kb region in the E. coli draft assembly. To see how we generated these files please refer to the tutorial creating_example_dataset.

You should find the following files:

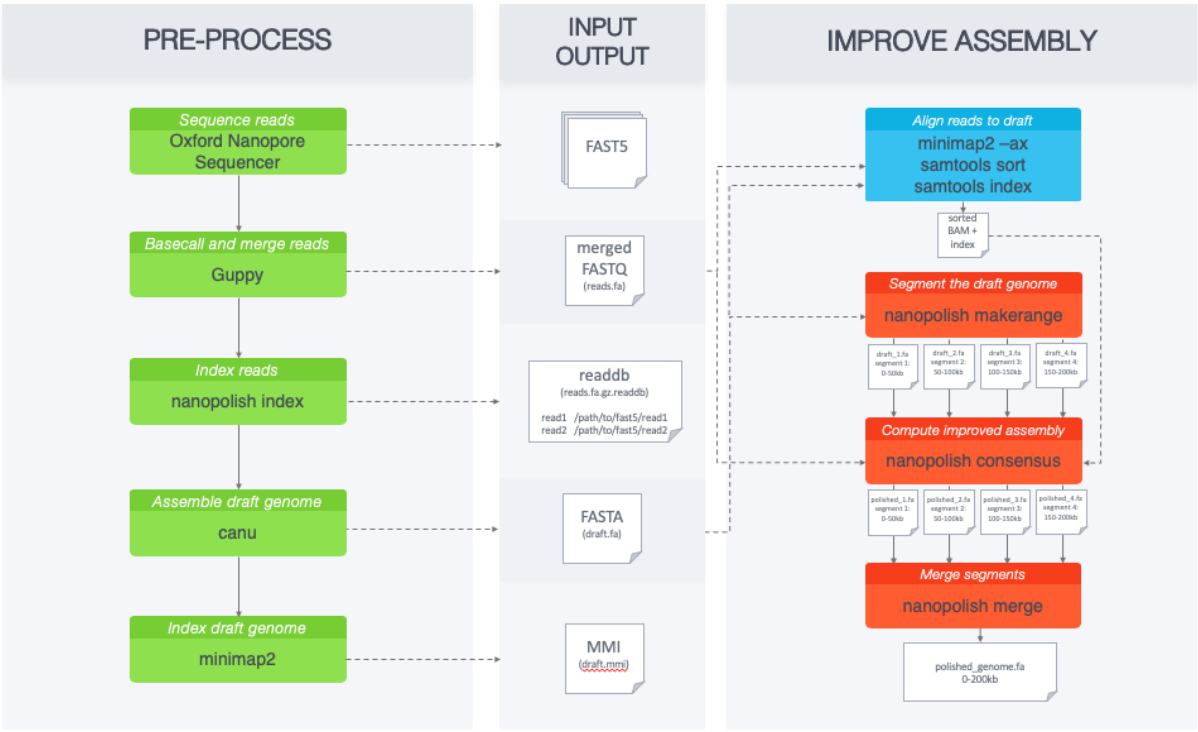
- `reads.fasta` : subset of basecalled reads
- `draft.fa` : draft genome assembly
- `draft.fa.fai` : draft genome assembly index
- `fast5_files/` : a directory containing FAST5 files
- `ecoli_2kb_region.log` : a log file for how the dataset was created with nanopolish helper script (`scripts/extract_reads_aligned_to_region.py`)

For the evaluation step you will need the reference genome:

```
curl -o ref.fa https://ftp.ncbi.nih.gov/genomes/archive/old_genbank/Bacteria/
↪Escherichia_coli_K_12_substr__MG1655_uid225/U00096.ffn
```

2.2 Analysis workflow

The pipeline below describes the recommended analysis workflow for larger datasets. In this tutorial, we will run through the basic steps of the pipeline for this smaller (2kb) dataset.



2.3 Data preprocessing

nanopolish needs access to the signal-level data measured by the nanopore sequencer. To begin, we need to create an index `readdb` file that links read ids with their signal-level data in the FAST5 files:

```
nanopolish index -d fast5_files/ reads.fasta
```

We get the following files: `reads.fasta.index`, `reads.fasta.index.fai`, `reads.fasta.index.gzi`, and `reads.fasta.index.readdb`.

2.4 Compute the draft genome assembly using canu

As computing the draft genome assembly takes a few hours we have included the pre-assembled data for you (`draft.fa`). We used the following parameters with canu:

```
canu \
    -p ecoli -d outdir genomeSize=4.6m \
    -nanopore-raw albacore-2.0.1-merged.fastq
```

2.5 Compute a new consensus sequence for a draft assembly

Now that we have `reads.fasta` indexed with `nanopolish index`, and have a draft genome assembly `draft.fa`, we can begin to improve the assembly with nanopolish. Let us get started!

First, we align the original reads (`reads.fasta`) to the draft assembly (`draft.fa`) and sort alignments:

```
minimap2 -ax map-ont -t 8 draft.fa reads.fasta | samtools sort -o reads.sorted.bam -T␣
↪reads.tmp
samtools index reads.sorted.bam
```

Checkpoint: we can do a quick check to see if this step worked. The bam file should not be empty.

```
samtools view reads.sorted.bam | head
```

Then we run the consensus algorithm. For larger datasets we use `nanopolish_makerange.py` to split the draft genome assembly into 50kb segments, so that we can run the consensus algorithm on each segment in parallel. The output would be the polished segments in `fasta` format. Since our dataset is only covering a 2kb region, we skip this step and use the following command:

```
nanopolish variants --consensus -o polished.vcf \
    -w "tig00000001:200000-202000" \
    -r reads.fasta \
    -b reads.sorted.bam \
    -g draft.fa
```

We are left with: `polished.vcf`.

Note: As of v0.10.1, `nanopolish variants --consensus` only outputs a VCF file instead of a fasta sequence.

To generate the polished genome in fasta format:

```
nanopolish vcf2fasta --skip-checks -g draft.fa polished.vcf > polished_genome.fa
```

We only polished a 2kb region, so let's pull that out:

```
samtools faidx polished_genome.fa
samtools faidx polished_genome.fa "tig00000001:200000-202000" > polished.fa
```

2.6 Evaluate the assembly

To analyze how nanopolish performed improving the accuracy we use MUMmer. MUMmer contains "dnadiff", a program that enables us to see a report on alignment statistics. With dnadiff we can compare the two different assemblies.

```
mkdir analysis
MUMmer3.23/dnadiff --prefix analysis/draft.dnadiff ref.fa draft.fa
MUMmer3.23/dnadiff --prefix analysis/polished.dnadiff ref.fa polished.fa
```

This generates `draft.dnadiff.report` and `polished.dnadiff.report` along with other files. The metric we are interested in is `AvgIdentity` under `[ Alignments ] 1-to-1`, which is a measurement of how similar the genome assemblies are to the reference genome. We expect to see a higher value for the polished assembly than the draft (`99.90` vs `99.53`), concluding that the nanopolish consensus algorithm worked successfully.

Note: The example dataset was PCR amplified causing a loss of methylation information. We recommend using the `-q dam,dcm` with `nanopolish variants --consensus` if you have data with methylation information to account for known bacterial methyltransferases.

CHAPTER 3

Quickstart - how to align events to a reference genome

The eventalign module in nanopolish is used to align events or "squiggles" to a reference genome. We (the developers of nanopolish) use this feature extensively when we want to see what the low-level signal information looks like. It helps us model the signal and discover differences in current that might hint at base modifications. Here we provide a step-by-step tutorial to help you get started with the nanopolish eventalign module.

For more information about eventalign:

- Blog post: "Aligning Nanopore Events to a Reference"
- Paper: "A complete bacterial genome assembled de novo using only nanopore sequencing data"

Requirements:

- nanopolish
- samtools
- minimap2

3.1 Download example dataset

You can download the example dataset we will use here:

```
wget http://s3.climb.ac.uk/nanopolish_tutorial/ecoli_2kb_region.tar.gz
tar -xvf ecoli_2kb_region.tar.gz
cd ecoli_2kb_region
```

Details:

- Sample : E. coli str. K-12 substr. MG1655
- Instrument : MinION sequencing R9.4 chemistry
- Basecaller : Albacore v2.0.1
- Region: "tig00000001:200000-202000"

- Note: Ligation-mediated PCR amplification performed

This is a subset of reads that aligned to a 2kb region in the E. coli draft assembly. To see how we generated these files please refer to this section: *Tutorial - using extraction helper script to create example datsets*.

You should find the following files:

- `reads.fasta` : subset of basecalled reads
- `fast5_files/` : a directory containing FAST5 files

You will need the E. coli reference genome:

```
curl -o ref.fa https://ftp.ncbi.nih.gov/genomes/archive/old_genbank/Bacteria/
↪Escherichia_coli_K_12_substr__MG1655_uid225/U00096.ffn
```

3.2 Align the reads with minimap2

We first need to index the reads:

```
nanopolish index -d fast5_files/ reads.fasta
```

Output files: `reads.fasta.index`, `reads.fasta.index.fai`, `reads.fasta.index.gzi`, and `reads.fasta.index.readdb`.

Then we can align the reads to the reference:

```
minimap2 -ax map-ont -t 8 ref.fa reads.fasta | samtools sort -o reads-ref.sorted.bam -
↪T reads.tmp
samtools index reads-ref.sorted.bam
```

Output files: `reads-ref.sorted.bam` and `reads-ref.sorted.bam.bai`.

Checkpoint: Let's see if the bam file is not truncated. This will check that the beginning of the file contains a valid header, and checks if the EOF is present. This will exit with a non-zero exit code if the conditions were not met:

```
samtools quickcheck reads-ref.sorted.bam
```

3.3 Align the nanopore events to a reference

Now we are ready to run nanopolish to align the events to the reference genome:

```
nanopolish eventalign \
    --reads reads.fasta \
    --bam reads-ref.sorted.bam \
    --genome ref.fa \
    --scale-events > reads-ref.eventalign.txt
```

3.4 Assess the eventalign output

If we take a peek at the first few lines of `reads-ref.eventalign.txt` this is what we get:

```
contig    position  reference_kmer  read_index  strand  event_index  event_level_mean␣
↪ event_stdv  event_length  model_kmer  model_mean  model_stdv  standardized_level
gi|545778205|gb|U00096.3|:c514859-514401  3         ATGGAG          0           t
↪  16538        89.82             3.746       0.00100       CTCCAT       92.53
↪2.49        -0.88
gi|545778205|gb|U00096.3|:c514859-514401  3         ATGGAG          0           t
↪  16537        88.89             2.185       0.00100       CTCCAT       92.53
↪2.49        -1.18
gi|545778205|gb|U00096.3|:c514859-514401  3         ATGGAG          0           t
↪  16536        94.96             2.441       0.00125       CTCCAT       92.53
↪2.49        0.79
gi|545778205|gb|U00096.3|:c514859-514401  3         ATGGAG          0           t
↪  16535        81.63             2.760       0.00150       NNNNNN       0.00
↪0.00        inf
gi|545778205|gb|U00096.3|:c514859-514401  7         AGTTAA          0           t
↪  16534        78.96             2.278       0.00075       TTAACT       75.55
↪3.52        0.79
gi|545778205|gb|U00096.3|:c514859-514401  8         GTTAAT          0           t
↪  16533        98.81             4.001       0.00100       ATTAAC       95.87
↪3.30        0.72
gi|545778205|gb|U00096.3|:c514859-514401  9         TTAATG          0           t
↪  16532        96.92             1.506       0.00150       CATTAA       95.43
↪3.32        0.36
gi|545778205|gb|U00096.3|:c514859-514401  10        TAATGG          0           t
↪  16531        70.86             0.402       0.00100       CCATTA       68.99
↪3.70        0.41
gi|545778205|gb|U00096.3|:c514859-514401  11        AATGGT          0           t
↪  16530        91.24             4.256       0.00175       ACCATT       85.84
↪2.74        1.60
```

3.5 Example plots

In Figure 1 of our methylation detection paper we show a histogram of `event_level_mean` for a selection of k-mers to demonstrate how methylation changes the observed current. The data for these figures was generated by eventalign, which we subsequently plotted in R using ggplot2.

CHAPTER 4

Quickstart - calling methylation with nanopolish

Oxford Nanopore sequencers are sensitive to base modifications. Here we provide a step-by-step tutorial to help you get started with detecting base modifications using nanopolish.

For more information about our approach:

- Simpson, Jared T., et al. "Detecting DNA cytosine methylation using nanopore sequencing." Nature Methods (2017).

Requirements:

- nanopolish v0.8.4
- samtools v1.2
- minimap2

4.1 Download example dataset

In this tutorial we will use a subset of the NA12878 WGS Consortium data. You can download the example dataset we will use here (warning: the file is about 2GB):

```
wget http://s3.climb.ac.uk/nanopolish_tutorial/methylation_example.tar.gz
tar -xvf methylation_example.tar.gz
cd methylation_example
```

Details:

- Sample : Human cell line (NA12878)
- Basecaller : Guppy v2.3.5
- Region: chr20:5,000,000-10,000,000

In the extracted example data you should find the following files:

- `albacore_output.fastq` : the subset of the basecalled reads

- `reference.fasta` : the chromsome 20 reference sequence
- `fast5_files/` : a directory containing signal-level FAST5 files

The reads were basecalled using this albacore command:

```
guppy_basecaller -c dna_r9.4.1_450bps_flipflop.cfg -i fast5_files/ -s basecalled/
```

After the basecaller finished, we merged all of the fastq files together into a single file:

```
cat basecalled/*.fastq > output.fastq
```

4.2 Data preprocessing

nanopolish needs access to the signal-level data measured by the nanopore sequencer. To begin, we need to create an index file that links read ids with their signal-level data in the FAST5 files:

```
nanopolish index -d fast5_files/ output.fastq
```

We get the following files: `output.fastq.index`, `output.fastq.index.fai`, `output.fastq.fastq.index.gzi`, and `output.fastq.index.readdb`.

4.3 Aligning reads to the reference genome

Next, we need to align the basecalled reads to the reference genome. We use minimap2 as it is fast enough to map reads to the human genome. In this example we'll pipe the output directly into `samtools sort` to get a sorted bam file:

```
minimap2 -a -x map-ont reference.fasta output.fastq | samtools sort -T tmp -o output.
↪sorted.bam
samtools index output.sorted.bam
```

4.4 Calling methylation

Now we're ready to use nanopolish to detect methylated bases (in this case 5-methylcytosine in a CpG context). The command is fairly straightforward - we have to tell it what reads to use (`output.fastq`), where the alignments are (`output.sorted.bam`), the reference genome (`reference.fasta`) and what region of the genome we're interested in (`chr20:5,000,000-10,000,000`):

```
nanopolish call-methylation -t 8 -r output.fastq -b output.sorted.bam -g reference.
↪fasta -w "chr20:5,000,000-10,000,000" > methylation_calls.tsv
```

The output file contains a lot of information including the position of the CG dinucleotide on the reference genome, the ID of the read that was used to make the call, and the log-likelihood ratio calculated by our model:

```
chromosome  start    end      read_name                            log_lik_ratio␣
↪log_lik_methylated  log_lik_unmethylated  num_calling_strands  num_cpgs  sequence
chr20       4980553  4980553  c1e202f4-e8f9-4eb8-b9a6-d79e6fab1e9a  3.70          -
↪167.47              -171.17               1                    1         TGAGACGGGGT
chr20       4980599  4980599  c1e202f4-e8f9-4eb8-b9a6-d79e6fab1e9a  2.64          -
↪98.87               -101.51               1                    1         AATCTCGGCTC
```

(continues on next page)

(continued from previous page)

```
chr20       4980616  4980616  c1e202f4-e8f9-4eb8-b9a6-d79e6fab1e9a  -0.61          -
↪95.35              -94.75               1                    1         ACCTCCGCCTC
chr20       4980690  4980690  c1e202f4-e8f9-4eb8-b9a6-d79e6fab1e9a  -2.99          -
↪99.58              -96.59               1                    1         ACACCCGGCTA
chr20       4980780  4980780  c1e202f4-e8f9-4eb8-b9a6-d79e6fab1e9a  5.27           -
↪135.45             -140.72              1                    1         CACCTCGGCCT
chr20       4980807  4980807  c1e202f4-e8f9-4eb8-b9a6-d79e6fab1e9a  -2.95          -
↪89.20              -86.26               1                    1         ATTACCGGTGT
chr20       4980820  4980822  c1e202f4-e8f9-4eb8-b9a6-d79e6fab1e9a  7.47           -
↪90.63              -98.10               1                    2         ⌴
↪GCCACCGCGCCCA
chr20       4980899  4980901  c1e202f4-e8f9-4eb8-b9a6-d79e6fab1e9a  3.17           -
↪96.40              -99.57               1                    2         ⌴
↪GTATACGCGTTCC
chr20       4980955  4980955  c1e202f4-e8f9-4eb8-b9a6-d79e6fab1e9a  0.33           -
↪92.14              -92.47               1                    1         AGTCCCGATAT
```

A positive value in the `log_lik_ratio` column indicates support for methylation. We have provided a helper script that can be used to calculate how often each reference position was methylated:

```
scripts/calculate_methylation_frequency.py methylation_calls.tsv > methylation_
↪frequency.tsv
```

The output is another tab-separated file, this time summarized by genomic position:

```
chromosome  start    end      num_cpgs_in_group  called_sites  called_sites_
↪methylated  methylated_frequency  group_sequence
chr20       5036763  5036763  1                  21            20
↪  0.952                 split-group
chr20       5036770  5036770  1                  21            20
↪  0.952                 split-group
chr20       5036780  5036780  1                  21            20
↪  0.952                 split-group
chr20       5037173  5037173  1                  13            5
↪  0.385                 AAGGACGTTAT
```

In the example data set we have also included bisulfite data from ENCODE for the same region of chromosome 20. We can use the included `compare_methylation.py` helper script to do a quick comparison between the nanopolish methylation output and bisulfite:

```
python3 compare_methylation.py bisulfite.ENCFF835NTC.example.tsv methylation_
↪frequency.tsv > bisulfite_vs_nanopolish.tsv
```

We can use R to visualize the results - we observe good correlation between the nanopolish methylation calls and bisulfite:

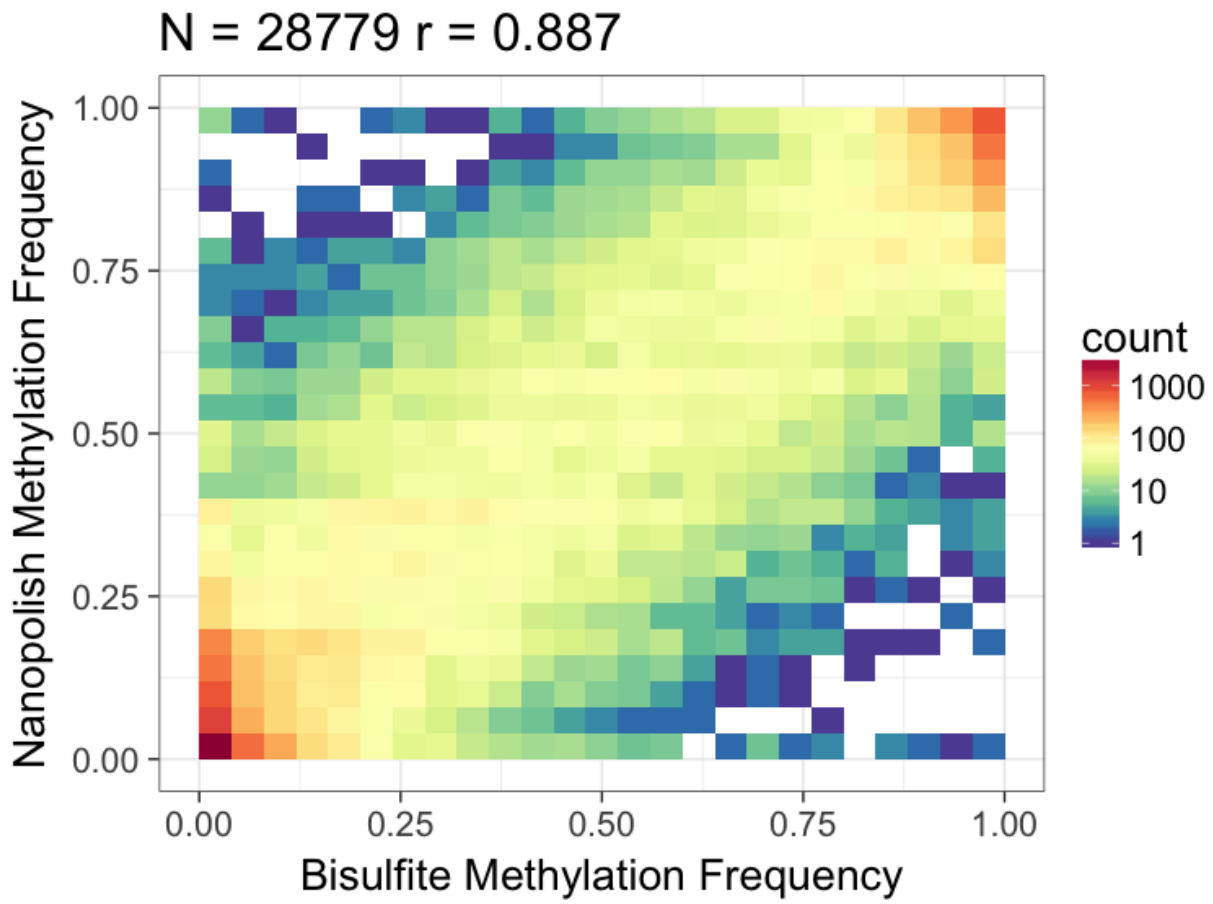
```
library(ggplot2)
library(RColorBrewer)
data <- read.table("bisulfite_vs_nanopolish.tsv", header=T)

# Set color palette for 2D heatmap
rf <- colorRampPalette(rev(brewer.pal(11,'Spectral')))
r <- rf(32)

c <- cor(data$frequency_1, data$frequency_2)
title <- sprintf("N = %d r = %.3f", nrow(data), c)
```

(continues on next page)

(continued from previous page)

```
ggplot(data, aes(frequency_1, frequency_2)) +
    geom_bin2d(bins=25) + scale_fill_gradientn(colors=r, trans="log10") +
    xlab("Bisulfite Methylation Frequency") +
    ylab("Nanopolish Methylation Frequency") +
    theme_bw(base_size=20) +
    ggtitle(title)
```

Here's what the output should look like:



CHAPTER 5

Quickstart - how to estimate poly(A) tail lengths from nanopore native RNA reads

Owing to homopolymer effects and the proximity to the sequencing adapters, the polyadenylated tails of reads obtained from nanopore native RNA sequencing are improperly basecalled, making their lengths difficult to quantify. We developed the *polya* subprogram to use an alternative statistical model to estimate these tail lengths.

In this quickstart tutorial, we'll show you how to estimate polyadenylated tail lengths step-by-step, starting from nothing but raw fast5 files. We'll basecall the fast5 files with Oxford Nanopore Technologies' *albacore* basecaller, before aligning the resulting reads with *minimap2*, indexing the files with *nanopolish index*, and finally segmenting the reads and calling the tail lengths with *nanopolish polya*.

We'll be following the steps taken in our benchmarking analysis workflow that accompanies our publication. In each step below, we'll generate files in the working directory instead of making subdirectories, except in the case of large datasets.

5.1 Software requirements

- nanopolish >= *v10.2*
- minimap2 >= *v2.12*
- samtools >= *v1.9*
- *albacore* >= *v2.3.3* (from Oxford Nanopore's private customer portal)

5.2 Download raw fast5 data and basecall

Let's start by downloading a dataset of fast5 files from the European Nucleotide Archive. We'll download a tarball of fast5 files containing reads that are known to have polyadenylated tail lengths of 30nt.

```
mkdir data && mkdir data/fastqs
wget ftp://ftp.sra.ebi.ac.uk/vol1/run/ERR276/ERR2764784/30xpolyA.tar.gz -O 30xpolyA.
↪tar.gz && mv 30xpolyA.tar.gz data/
tar -xzf data/30xpolyA.tar.gz -C data/
guppy_basecaller -c rna_r9.4.1_70bps_hac.cfg -i data/30xpolyA/fast5/pass -s data/
↪fastqs
cat data/fastqs/*.fastq > data/30xpolyA.fastq
```

In the above, change the value of the *-f* and *-k* arguments based on your flow-cell and sequencing kit, as the basecaller's accuracy is highly dependent upon these settings.

Our directory structure should now look something like this:

```
(current directory)
|- data/
|-- 30xpolyA.fastq
|-- 30xpolyA.tar.gz
|-- 30xpolyA/
|--- fast5/
|---- pass/
|----- raw_read1.fast5
|----- (... more raw fast5's here ...)
|-- fastqs/
|--- workspace/
|---- pass/
```

5.3 Index with nanopolish index

Let's construct an index for our reads with nanopolish's *index* subprogram. This constructs a fast lookup table that tells our program where to find the raw fast5 file for each basecalled read.

```
nanopolish index --directory=data/30xpolyA/fast5/pass --sequencing-summary=data/
↪fastqs/sequencing_summary.txt data/30xpolyA.fastq
```

This should generate a collection of files in the same directory that contains the *30xpolyA.fastq*.

5.4 Align with minimap2 and format the BAM file

Now we'll align our basecalled mRNA sequences in the fastq file with our reference. First download a reference fasta:

```
wget https://raw.githubusercontent.com/paultsw/polya_analysis/master/data/references/
↪enolase_reference.fas
wget https://raw.githubusercontent.com/paultsw/polya_analysis/master/data/references/
↪enolase_reference.fas.fai
mv enolase_reference.* data/
```

Note that our directory structure should now look like this:

```
(current directory)
|- data/
|-- enolase_reference.fas
|-- enolase_reference.fai
|-- (... same structure as above ...)
```

Let's run *minimap2* to align our basecalled reads to this reference and generate a collection of SAM/BAM files with *samtools*:

```
minimap2 -a -x map-ont data/enolase_reference.fas data/30xpolyA.fastq | samtools view␣
↪-b - -o data/30xpolyA.bam
cd data && samtools sort -T tmp -o 30xpolyA.sorted.bam 30xpolyA.bam && samtools index␣
↪30xpolyA.sorted.bam && cd ..
```

Note that we used *-x map-ont*, which is typically for unspliced reads (e.g. genomic DNA) coming from nanopore sequencers. In typical native mRNA sequencing, you should use *-x splice*, which is a splice-aware setting (and uses a different gap cost in the alignment). We're using *map-ont* due to the fact that our reads come from a control dataset with no splicing.

There should be three more files in the *data* directory now: *30xpolyA.bam*, *30xpolyA.sorted.bam*, and *30xpolyA.sorted.bam.bai*.

5.5 Segmentation and tail length estimation with nanopolish polya

Finally, we can run the polyadenylation estimator:

```
nanopolish polya --threads=8 --reads=data/30xpolyA.fastq --bam=data/30xpolyA.sorted.
↪bam --genome=data/enolase_reference.fas > polya_results.tsv
```

Set the *–threads* flag to the number of parallel threads you want to use. Generally speaking, a larger number of threads tends to lower the compute time, but there are diminishing returns to a higher value and performance can actually decrease if your CPU is incapable of supporting your desired number of parallel threads. The best number of threads to use is highly dependent upon your hardware.

5.6 Interpreting the output TSV file

We'll end this quickstart with a look at the output of the polya program. Let's look at the top five lines of the *polya_results.tsv* file we've just generated:

```
head -20 polya_results.tsv | column -t
readname                              contig   position  leader_start  adapter_start␣
↪polya_start  transcript_start  read_rate  polya_length  qc_tag
d6f42b79-90c6-4edd-8c8f-8a7ce0ac6ecb  YHR174W  0         54.0          3446.0␣
↪7216.0       8211.0            130.96     38.22         PASS
453f3f3e-d22f-4d9c-81a6-8576e23390ed  YHR174W  0         228.0         5542.0␣
↪10298.0      11046.0           130.96     27.48         PASS
e02d9858-0c04-4d86-8dba-18a47d9ac005  YHR174W  0         221.0         1812.0␣
↪7715.0       8775.0            97.16      29.16         PASS
b588dee2-2c5b-410c-91e1-fe8140f4f837  YHR174W  0         22.0          8338.0␣
↪13432.0      14294.0           130.96     32.43         PASS
af9dfee2-1711-4083-b109-487b99895e0a  YHR174W  0         889.0         3679.0␣
↪6140.0       7168.0            130.96     39.65         PASS
93f98a86-3b18-48cf-8c4d-15cf277911e2  YHR174W  0         144.0         1464.0␣
↪5615.0       6515.0            120.48     30.96         SUFFCLIP
af9dfee2-1711-4083-b109-487b99895e0a  YHR174W  0         889.0         3679.0␣
↪6140.0       7168.0            130.96     39.65         SUFFCLIP
93f98a86-3b18-48cf-8c4d-15cf277911e2  YHR174W  0         144.0         1464.0␣
↪5615.0       6515.0            120.48     30.96         PASS
ca8d4059-9d82-45ee-aa07-4b8b351618b3  YHR174W  0         1.0           2157.0␣
↪4255.0       5862.0            111.56     54.48         PASS
```

(continues on next page)

(continued from previous page)

```
3493c123-78d4-4f7c-add0-cbb249aef00a  YHR174W  0  78.0  1938.0  ⌐
↪4829.0  5491.0  136.91  25.05  PASS
f5ff1802-3fdd-479a-8888-c72de01bbaea  YHR174W  0  150.0  3476.0  ⌐
↪7233.0  7932.0  130.96  25.35  PASS
bb929728-2ed8-42b0-a5a5-eea4bfd62673  YHR174W  0  91.0  1061.0  ⌐
↪6241.0  6910.0  111.56  19.74  PASS
17cf3fef-1acb-4045-8252-e9c00fedfb7c  YHR174W  0  447.0  6004.0  ⌐
↪20058.0  20964.0  100.40  25.17  ADAPTER
e3e38de6-8a99-4029-a067-261f470517ca  YHR174W  0  41.0  1588.0  ⌐
↪4057.0  5303.0  130.96  49.13  PASS
66f55b56-c22e-4e6d-999e-50687bed6fb7  YHR174W  0  191.0  3160.0  ⌐
↪9218.0  10030.0  125.50  28.79  PASS
56c116d7-9286-4b57-8329-e74928b11b38  YHR174W  0  13.0  1516.0  ⌐
↪5845.0  6773.0  130.96  35.30  PASS
5ca1392c-c48f-4135-85d3-271bd4ee7a13  YHR174W  0  1.0  1976.0  ⌐
↪4854.0  5947.0  136.91  44.64  PASS
66b5a0ef-b8e6-475e-bf20-04b96154a67f  YHR174W  0  98.0  3847.0  ⌐
↪7066.0  7925.0  120.48  29.32  PASS
34bf2187-5816-4744-8e6a-3250b5247e02  YHR174W  0  1.0  2897.0  ⌐
↪6885.0  7547.0  125.50  22.54  PASS
```

Each row corresponds to the output for a given read. The columns have the following interpretation: * *readname* refers to the unique ID associated to this particular read. This string is also used to look up the corresponding fast5 file, e.g. by looking

through the *readdb* file generated by *nanopolish index*.

- *contig* refers to the reference sequence that this read aligns to, and is taken from the BAM file.
- *position* is the 5' starting position of the alignment to the reference sequence, and also comes from the BAM file.
- *leader_start*, *adapter_start*, *polya_start*, and *transcript_start* are the indices of the signal segmentation generated by the underlying model within *nanopolish*. Briefly, there are four biologically-meaningful regions of the raw sequence of electrical current readings within each fast5 file; these four entries denote the starting index of each of these consecutive regions. The indices start from 0 and are oriented in the 3'-to-5' direction (due to the inherent orientation of the native RNA nanopore sequencing protocol). A full exposition of this segmentation algorithm is available in the **'supplementary note<https://www.biorxiv.org/content/biorxiv/suppl/2018/11/09/459529.DC1/459529-2.pdf>'_** to our associated publication.
- *read_rate* is the estimated translocation rate (in units of nucleotides/second) of the read through the pore. The translocation rate is non-uniform during the sequencing process of even a single molecule, so this is ultimately a summary statistic of the dynamic, time-varying rate.
- *polya_length* is the estimated polyadenylated tail length, in number of nucleotides. That this value is a float rather than an integer reflects the fact that our estimated tail length is the output of an estimator based on the translocation rate.
- *qc_tag* is an additional flag used to indicate the validity of the estimate. Generally speaking, you should only use rows of the output file with this value set to *PASS*; all other rows with (e.g.) the *qc_tag* set to *SUFFCLIP*, *ADAPTER*, etc. display signs of irregularity that indicate that we believe the estimate to be unreliable. You can easily filter away all rows with the tag set to anything other than *PASS* using a *grep*:

```
grep 'PASS' polya_results.tsv > polya_results.pass_only.tsv
```

CHAPTER 6

Helping us debug nanopolish

6.1 Overview

Running into errors with nanopolish? To help us debug, we need to be able to reproduce the errors. We can do this by packaging a subset of the files that were used by a nanopolish. We have provided `scripts/extract_reads_aligned_to_region.py` and this tutorial to help you do exactly this.

Briefly, this script will:

- extract reads that align to a given region in the draft genome assembly
- rewrite a new BAM, BAI, FASTA file with these reads
- extract the FAST5 files associated with these reads
- save all these files into a tar.gz file

6.2 Workflow

1. Narrow down a problematic region by running `nanopolish variants --consensus [...]` and changing the `-w` parameter.
2. Run the `scripts/extract_reads_aligned_to_region.py`.
3. Send the resulting `tar.gz` file to us by hosting either a dropbox or google drive.

6.3 Tutorial - using extraction helper script to create example datsets

We extracted a subset of reads for a 2kb region to create the example dataset for the eventalign and consensus tutorial using `scripts/extract_reads_aligned_to_region.py`. Here is how:

Generated basecalled `--reads` file:

1. Basecalled reads with albacore:

```
read_fast5_basecaller.py -c r94_450bps_linear.cfg -t 8 -i /path/to/raw/fast5/
↪files -s /path/to/albacore-2.0.1/output/directory -o fastq
```

2. Merged the different albacore fastq outputs:

```
cat /path/to/albacore-2.0.1/output/directory/workspace/pass/*.fastq > albacore-2.
↪0.1-merged.fastq
```

3. Converted the merged fastq to fasta format:

```
paste - - - - < albacore-2.0.1-merged.fastq | cut -f 1,2 | sed 's/^@/>/' | tr "\t
↪" "\n" > reads.fasta
```

Generated `--bam` file with the draft genome assembly (`-g`):

1. Ran canu to create draft genome assembly:

```
canu \
    -p ecoli -d outdir genomeSize=4.6m \
    -nanopore-raw reads.fasta \
```

2. Index draft assembly:

```
bwa index ecoli.contigs.fasta
samtools faidx ecoli.contigs.fasta
```

3. Aligned reads to draft genome assembly with bwa (v0.7.12):

```
bwa mem -x ont2d ecoli.contigs.fasta reads.fasta | samtools sort -o reads.sorted.
↪bam -T reads.tmp
samtools index reads.sorted.bam
```

Selected a `--window`:

1. Identified the first contig name and chose a random start position:

```
head -3 ecoli.contigs.fasta
```

Output:

```
>tig00000001 len=4376233 reads=23096 covStat=7751.73 gappedBases=no class=contig␣
↪suggestRepeat=no suggestCircular=no
AGATGCTTTGAAAGAAACGCAGAATAGATCTCTATGTAATGATATGGAATACTCTGGTATTGTCTGTAAAGATACTAATGGAAAATATTTTGCATCTAAG
GCAGAAACTGATAATTTAAGAAAGGAGTCATATCCTCTGAAAAGAAAATGTCCCACAGGTACAGATAGAGTTGCTGCTTATCATACTCACGGTGCAGATA
```

As we wanted a 2kb region, we selected a random start position (200000) so our end position was 202000. Therefore our `--window` was "tig00000001:200000-202000".

Using the files we created, we ran `scripts/extract_reads_aligned_to_region.py`, please see usage example below.

Note: Make sure nanopolish still reproduces the same error on this subset before sending it to us.

6.4 Usage example

```
python3 extract_reads_aligned_to_region.py \
    --reads reads.fasta \
    --genome ecoli.contigs.fasta \
    --bam reads.sorted.bam \
    --window "tig00000001:200000-202000" \
    --output_prefix ecoli_2kb_region --verbose
```

Argument name(s)	Req.	Default value	Description
`-b, --bam`	Y	NA	Sorted bam file created by aligning reads to the draft genome.
`-g, --genome`	Y	NA	Draft genome assembly
`-r, --reads`	Y	NA	Fasta, fastq, fasta.gz, or fastq.gz file containing base-called reads.
`-w, --window`	Y	NA	Draft genome assembly coordinate string ex. "contig:start-end". It is essential that you wrap the coordinates in quotation marks (").
`-o, --output_prefix`	N	reads_subset	Prefix of output tar.gz and log file.
`-v, --verbose`	N	False	Use for verbose output with info on progress.

6.5 Script overview

1. Parse input files
2. Assumes readdb file name from input reads file
3. **Validates input**
    - checks that input bam, readdb, fasta/q, draft genome assembly, draft genome assembly index file exist, are not empy, and are readable

4. **With user input draft genome assembly coordinates, extracts all reads that aligned within these coordinates stores the rea**
    - uses pysam.AlignmentFile
    - uses samfile.fetch(region=draft_ga_coords) to get all reads aligned to region
    - if reads map to multiple sections within draft ga it is not added again
5. **Parses through the input readdb file to find the FAST5 files associated with each region that aligned to region**
    - stores in dictionary region_fast5_files; key = read_id, value = path/to/fast5/file
    - path to fast5 file is currently dependent on the user's directory structure
6. **Make a BAM and BAI file for this specific region**
    - creates a new BAM file called `region.bam`
    - with pysam.view we rewrite the new bam with reads that aligned to the region...
    - with pysam.index we create a new BAI file
7. **Now to make a new FASTA file with this subset of reads**
    - the new fasta file is called `region.fasta`
    - this first checks what type of sequences file is given { `fasta`, `fastq`, `fasta.gz`, `fastq.gz` }
    - then handles based on type of seq file using SeqIO.parse
    - then writes to a new fasta file
8. **Let's get to tarring**
    - creates a `tar.gz` file with the output prefix
    - saves the fast5 files in directory `output_prefix/fast5_files/`
9. Adds the new fasta, new bam, and new bai file with the subset of reads
10. Adds the draft genome asssembly and associated fai index file
11. **Performs a check**
    - the number of reads in the new BAM file, new FASTA file, and the number of files in the fast5_files directory should be equal
12. Outputs a `tar.gz` and `log` file. FIN!

CHAPTER 7

Manual

Modules available:

```
nanopolish extract: extract reads in FASTA or FASTQ format from a directory of FAST5␣
↪files
nanopolish call-methylation: predict genomic bases that may be methylated
nanopolish variants: detect SNPs and indels with respect to a reference genome
nanopolish variants --consensus: calculate an improved consensus sequence for a draft␣
↪genome assembly
nanopolish eventalign: align signal-level events to k-mers of a reference genome
nanopolish phase-reads: Phase reads using heterozygous SNVs with respect to a␣
↪reference genome
nanopolish polya: Estimate polyadenylated tail lengths on native RNA reads
```

7.1 extract

7.1.1 Overview

This module is used to extract reads in FASTA or FASTQ format from a directory of FAST5 files.

7.1.2 Input

- path to a directory of FAST5 files modified to contain basecall information

7.1.3 Output

- sequences of reads in FASTA or FASTQ format

7.1.4 Usage example

```
nanopolish extract [OPTIONS] <fast5|dir>
```

Argument name(s)	Required	Default value	Description
<fast5|dir>	Y	NA	FAST5 or path to directory of FAST5 files.
-r, --recurse	N	NA	Recurse into subdirectories
-q, --fastq	N	fasta format	Use when you want to extract to FASTQ format
-t, --type=TYPE	N	2d-or-template	The type of read either: {template, complement, 2d, 2d-or-template, any}
-b, --basecaller=NAME[:VERSION]	N	NA	consider only data produced by basecaller NAME, optionally with given exact VERSION
-o, --output=FILE	N	stdout	Write output to FILE

7.2 index

7.2.1 Overview

Build an index mapping from basecalled reads to the signals measured by the sequencer

7.2.2 Input

- path to directory of raw nanopore sequencing data in FAST5 format
- basecalled reads

7.2.3 Output

- gzipped FASTA file of basecalled reads (.index)
- index files (.fai, .gzi, .readdb)

7.2.4 Readdb file format

Readdb file is a tab-separated file that contains two columns. One column represents read ids and the other column represents the corresponding path to FAST5 file:

```
read_id_1    /path/to/fast5/containing/reads_id_1/signals
read_id_2    /path/to/fast5/containing/read_id_2/signals
```

7.2.5 Usage example

```
nanopolish index [OPTIONS] -d nanopore_raw_file_directory reads.fastq
```

Argument name(s)	Required	Default value	Description
-d, --directory	Y	NA	FAST5 or path to directory of FAST5 files containing ONT sequencing raw signal information.
-f, --fast5-fofn	N	NA	file containing the paths to each fast5 for the run

7.3 call-methylation

7.3.1 Overview

Classify nucleotides as methylated or not.

7.3.2 Input

- Basecalled ONT reads in FASTA format

7.3.3 Output

- tab-separated file containing per-read log-likelihood ratios

7.3.4 Usage example

```
nanopolish call-methylation [OPTIONS] <fast5|dir>
```

Argument name(s)	Required	Default value	Description
-r, --reads=FILE	Y	NA	the ONT reads are in fasta FILE
-b, --bam=FILE	Y	NA	the reads aligned to the genome assembly are in bam FILE
-g, --genome=FILE	Y	NA	the genome we are computing a consensus for is in FILE
-t, --threads=NUM	N	1	use NUM threads
--progress	N	NA	print out a progress message

7.4 variants

7.4.1 Overview

This module is used to call single nucleotide polymorphisms (SNPs) using a signal-level HMM.

7.4.2 Input

- basecalled reads
- alignment info
- genome assembly

7.4.3 Output

- VCF file

7.4.4 Usage example

```
nanopolish variants [OPTIONS] --reads reads.fa --bam alignments.bam --genome genome.fa
```

Argument name(s)	Required	Default value	Description
`--snps`	N	NA	use flag to only call SNPs
`--consensus=FILE`	N	NA	run in consensus calling mode and write polished sequence to FILE
`--fix-homopolymers`	N	NA	use flag to run the experimental homopolymer caller
`--faster`	N	NA	minimize compute time while slightly reducing consensus accuracy
`-w, --window=STR`	N	NA	find variants in window STR (format: <chromsome_name>:<start>-<end>)
`-r, --reads=FILE`	Y	NA	the ONT reads are in fasta FILE
`-b, --bam=FILE`	Y	NA	the reads aligned to the reference genome are in bam FILE
`-e, --event-bam=FILE`	Y	NA	the events aligned to the reference genome are in bam FILE
`-g, --genome=FILE`	Y	NA	the reference genome is in FILE
`-o, --outfile=FILE`	N	stdout	write result to FILE
`-t, --threads=NUM`	N	1	use NUM threads
`-m, --min-candidate-frequency=F`	N	0.2	extract candidate variants from the aligned reads when the variant frequency is at least F
`-d, --min-candidate-depth=D`	N	20	extract candidate variants from the aligned reads when the depth is at least D
`-x, --max-haplotypes=N`	N	1000	consider at most N haplotypes combinations
`--max-rounds=N`	N	50	perform N rounds of consensus sequence improvement
`-c, --candidates=VCF`	N	NA	read variants candidates from VCF, rather than discovering them from aligned reads
`-a, --alternative-basecalls-bam=FILE`	N	NA	if an alternative basecaller was used that does not output event annotations then use basecalled sequences from FILE. The signal-level events will still be taken from the -b bam
`--calculate-all-support`	N	NA	when making a call, also calculate the support of the 3 other possible bases
`--models-fofn=FILE`	N	NA	read alternatives k-mer models from FILE

7.5 event align

7.5.1 Overview

Align nanopore events to reference k-mers

7.5.2 Input

- basecalled reads
- alignment information

- assembled genome

7.5.3 Usage example

```
nanopolish eventalign [OPTIONS] --reads reads.fa --bam alignments.bam --genome genome.
↪fa
```

Argument name(s)	Required	Default value	Description
`--sam`	N	NA	use to write output in SAM format
`-w, --window=STR`	N	NA	Compute the consensus for window STR (format : ctg:start_id-end_id)
`-r, --reads=FILE`	Y	NA	the ONT reads are in fasta FILE
`-b, --bam=FILE`	Y	NA	the reads aligned to the genome assembly are in bam FILE
`-g, --genome=FILE`	Y	NA	the genome we are computing a consensus for is in FILE
`-t, --threads=NUM`	N	1	use NUM threads
`--scale-events`	N	NA	scale events to the model, rather than vice-versa
`--progress`	N	NA	print out a progress message
`-n, --print-read-names`	N	NA	print read names instead of indexes
`--summary=FILE`	N	NA	summarize the alignment of each read/strand in FILE
`--samples`	N	NA	write the raw samples for the event to the tsv output
`--models-fofn=FILE`	N	NA	read alternative k-mer models from FILE

7.6 phase-reads - (experimental)

7.6.1 Overview

Phase reads using heterozygous SNVs with respect to a reference genome

7.6.2 Input

- basecalled reads
- alignment information
- assembled genome
- variants (from nanopolish variants or from other sources eg. Illumina VCF)

7.6.3 Usage example

```
nanopolish phase-reads [OPTIONS] --reads reads.fa --bam alignments.bam --genome␣
↪genome.fa variants.vcf
```

7.7 polya

7.7.1 Overview

Estimate the number of nucleotides in the poly(A) tails of native RNA reads.

7.7.2 Input

- basecalled reads
- alignment information
- reference transcripts

7.7.3 Usage example

```
nanopolish polya [OPTIONS] --reads=reads.fa --bam=alignments.bam --genome=ref.fa
```

Argument name(s)	Required	Default value	Description
`-w, --window=STR`	N	NA	Compute only for reads aligning to window of reference STR (format : ctg:start_id-end_id)
`-r, --reads=FILE`	Y	NA	the FAST(A/Q) file of native RNA reads
`-b, --bam=FILE`	Y	NA	the BAM file of alignments between reads and the reference
`-g, --genome=FILE`	Y	NA	the reference transcripts
`-t, --threads=NUM`	N	1	use NUM threads
`-v, -vv`	N	NA	*-v* returns raw sample log-likelihoods, while *-vv* returns event durations

CHAPTER 8

Publications

- Loman, Nicholas J., Joshua Quick, and Jared T. Simpson. "A complete bacterial genome assembled de novo using only nanopore sequencing data." Nature methods 12.8 (2015): 733-735.
- Quick, Joshua, et al. "Real-time, portable genome sequencing for Ebola surveillance." Nature 530.7589 (2016): 228-232.
- Simpson, Jared T., et al. "Detecting DNA cytosine methylation using nanopore sequencing." nature methods 14.4 (2017): 407-410.

CHAPTER 9

Credits and Thanks

The fast table-driven logsum implementation was provided by Sean Eddy as public domain code. This code was originally part of hmmer3 . Nanopolish also includes code from Oxford Nanopore's scrappie basecaller. This code is licensed under the MPL.