Processing nanopore reads with dada2

Preprocessing reads

Preparing reads

Dorado doesn't support demultiplexing of dual indexes on 5' and 3' ends. Additionally, ligated library reads can be either orientation. Most straightforward approach to solve this, is utilising flanking adapter sequences and cutadapt. Another alternative is to demultiplex index pairs in forward and reverse orientation, then build scripts to reverse complement and merge reverse reads.

Extracting forward reads to fastq file can be performed with following command

cutadapt -g t2f...rc(t2r) -O 13 –trimmed-only -m 1300 -M 1650 -o forward_out.fastq.gz raw_reads.fastq.gz

Extracting reverse reads

cutadapt -g t2r...rc(t2f) -O 13 -m 1300 -M 1650 –trimmed-only -o reverse_out.fastq.gz raw_reads.fastq.gz

Note! O, e, m and M parameters can be used to reduce chances of misaligned matches

Next step is to reverse complement reverse read file and join all reads to single file. This will make life easier later in the workflow.

seqkit seq -rp –seq-type DNA reverse_out.fastq.gz | gzip >reverse_comp.fastq.gz

Final step is to merge two files together

cat forward_out.fastq.gz reverse_comp.fastq.gz >nanopore_reads.fastq.gz

Demultiplexing reads

Prepare list of barcodes as a fasta file

Use cutadapt to demux fastq file. In example output files are written to demuxed subdirectory

cutadapt -e 0 -O 12 -g file:barcodes.fasta -o "demuxed/{name}.fastq.gz" input.fastq.gz

Trimming PCR amplification primers

You can use cutadapt and bash scripts to trim forward and revers PCR primers from demultiplexed sequence files.

Importing set1

Loading libraries

```
library(dada2);packageVersion("dada2")
```

```
[1] '1.30.0'
```

```
library(knitr);packageVersion("knitr")
```

```
[1] '1.45'
```

```
library(Biostrings);packageVersion("Biostrings")
```

```
[1] '2.70.2'
```

```
library(DECIPHER);packageVersion("DECIPHER")
```

```
[1] '2.30.0'
```

```
library(phyloseq);packageVersion("phyloseq")
```

```
[1] '1.46.0'
```

```
library(tidyverse);packageVersion("tidyverse")
```

```
[1] '2.0.0'
```

```
library(kableExtra);packageVersion("kableExtra")
```

```
[1] '1.4.0'
```

```
library(mia);packageVersion("mia")
```

```
[1] '1.10.0'
```

```
# Path variables
path <- "data/processed/set1"
training <- "~/feature_classifiers/SILVA_SSU_r138_2019.RData"
meta_file <- "data/set1_meta.tsv"
exportloc <- "results_set1/"
# Variables: truncation length, phix (Illumina)
truncation <- 1400
#Creates results directory
dir.create(exportloc)
#metadata
metadata <- data.frame(read_tsv(meta_file))
```

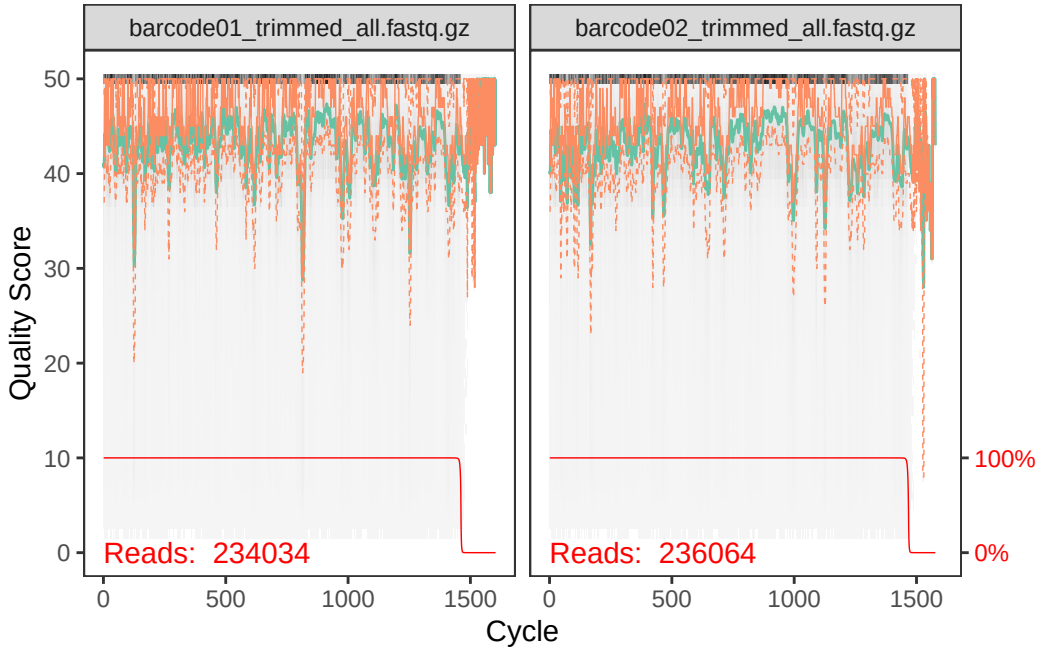
```r
#List files inside directory
list.files(path)
```

```
[1] "barcode01_trimmed_all.fastq.gz" "barcode02_trimmed_all.fastq.gz"
[3] "barcode03_trimmed_all.fastq.gz" "barcode04_trimmed_all.fastq.gz"
[5] "barcode05_trimmed_all.fastq.gz" "barcode06_trimmed_all.fastq.gz"
[7] "filtered"
```

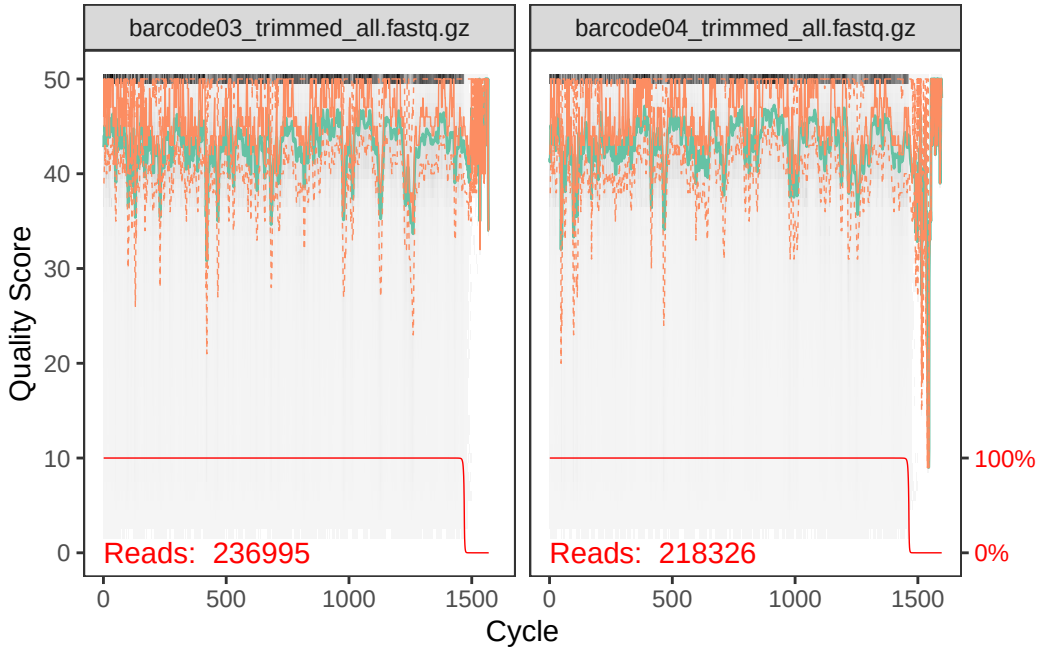
```r
# Forward fastq filenames have format: SAMPLENAME_R1_001.fastq
fnFs <- sort(list.files(path, pattern="_trimmed_all.fastq.gz", full.names = TRUE))
# Extract sample names, assuming filenames have format: SAMPLENAME_XXX.fastq
sample.names <- sapply(strsplit(basename(fnFs), "_"), `[`, 1)
```

Plotting sequence quality profile for sample pairs

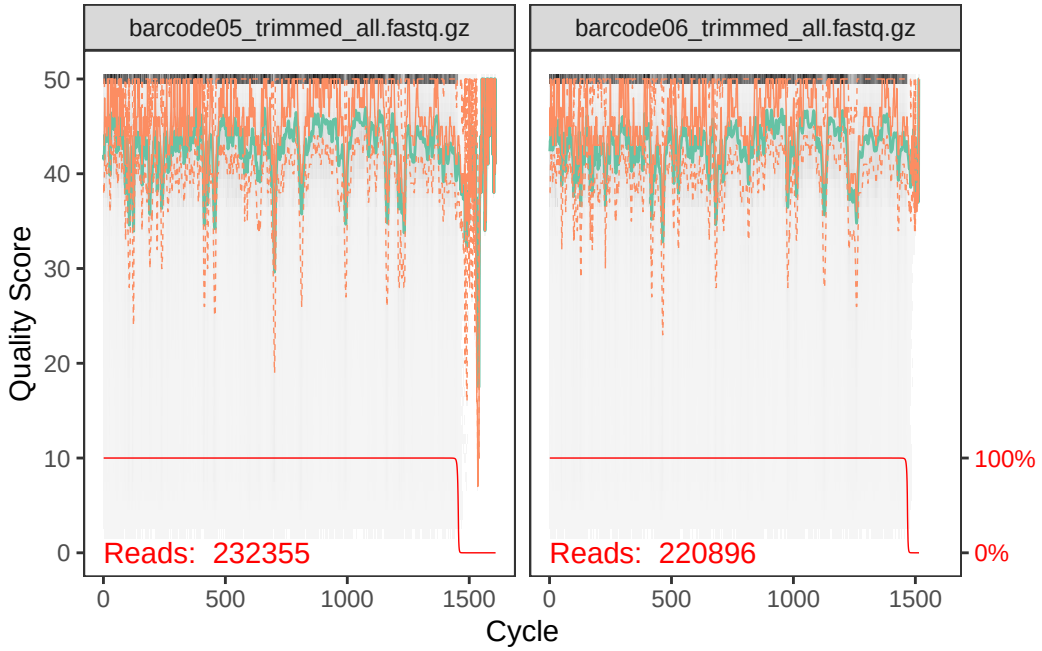
```
# Base quality plot
prI <- plotQualityProfile(fnFs[1:2])
prI
```



```
# Base quality plot
prII <- plotQualityProfile(fnFs[3:4])
prII
```



```
# Base quality plot
prIII <- plotQualityProfile(fnFs[5:6])
prIII
```



Denoising

Filter sequence data

Filtering reads (maxEE ≈ 1 error/200 bp sequence is good starting point for nanopore)

```
# Filtered files are placed in filtered subdirectory
filtFs <- file.path(path, "filtered", paste0(sample.names,
                                             "_F_filt.fastq.gz"))
# For single end data sets without phix control
names(filtFs) <- sample.names
out <- filterAndTrim(fnFs, filtFs, truncLen=truncation,
                     maxN = 0, maxEE = 7 , truncQ = 2,
                     compress = TRUE, multithread = TRUE,
                     rm.phix = FALSE)
```

Dereplicating sequences

Step condenses identical sequences saving computational time

```
derepFs <- derepFastq(filtFs, verbose = TRUE)
```

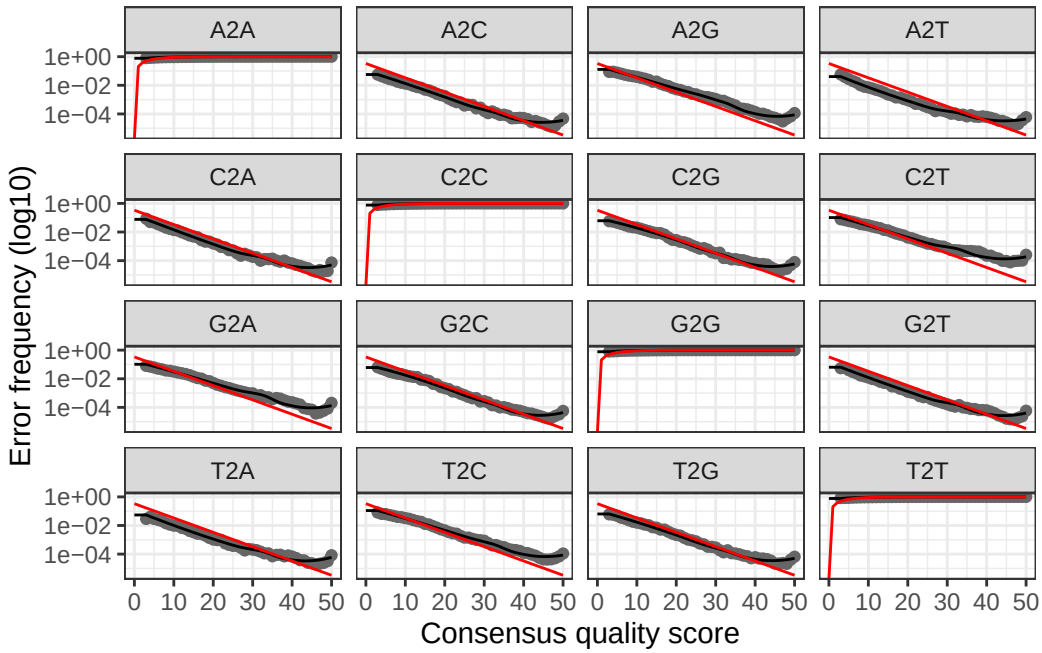
Learning error rates

```
# Forward read error rate
errF <- learnErrors(derepFs, multithread = TRUE)
```

```
269694600 total bases in 192639 reads from 1 samples will be used for learning the error rates.
```

Plotting error rate

```
# Plotting error rate profile for forward reads
plotErrors(errF, nominalQ = TRUE)
```



Denoising

```
dadaFs <- dada(derepFs, err = errF, multithread = TRUE)
```

```
Sample 1 - 192639 reads in 125108 unique sequences.
Sample 2 - 191813 reads in 146171 unique sequences.
Sample 3 - 188080 reads in 131856 unique sequences.
Sample 4 - 182492 reads in 128548 unique sequences.
Sample 5 - 174856 reads in 129654 unique sequences.
Sample 6 - 174343 reads in 126396 unique sequences.
```

Building asv table

```
seqtab <- makeSequenceTable(dadaFs)
# Dimensions of ASV table
dim(seqtab)
```

```
[1]  6 38
```

Chimera removal

```
seqtab.nochim <- removeBimeraDenovo(seqtab, method = "consensus",
                                    multithread = TRUE, verbose = TRUE)
dim(seqtab.nochim)
```

```
[1]  6 30
```

Summary

```
getN <- function(x) sum(getUniques(x))
track <- cbind(out, sapply(dadaFs, getN), rowSums(seqtab.nochim),
               rowSums(seqtab.nochim != 0))
#If processing a single sample, remove the sapply calls
colnames(track) <- c("Input", "Filtered", "DenoisedF", "Nonchimeric",
                     "N:o of variants")
rownames(track) <- metadata$Name
kable(track, caption="Summary table")  %>%
  kable_styling(latex_options=c("striped", "HOLD_position")) %>%
                row_spec(0,background = "teal", color = "ivory")
```

Table 1: Summary table

	Input	Filtered	DenoisedF	Nonchimeric	N:o of variants
PR01	234034	192639	192440	190867	7
PR02	236064	191813	191637	191637	6
PR03	236995	188080	187950	187950	3
PR04	218326	182492	182310	177817	5
PR05	232355	174856	174710	174710	4
PR06	220896	174343	174209	173858	5

Taxonomy assignment

IdTaxa from DECIPHER package

```
#Create a DNAStringSet from the ASV sequences
repseq <- DNAStringSet(getSequences(seqtab.nochim))
# CHANGE TO THE PATH OF YOUR TRAINING SET
load(training)
ids <- IdTaxa(repseq, trainingSet, strand = "top",
              processors = 3, verbose = FALSE,
              threshold = 50)
ranks <- c("domain", "phylum", "class", "order", "family",
           "genus", "species")
# Convert the output to a matrix analogous to the output from assignTaxonomy
taxid <-t(sapply(ids, function(x) {
        m <- match(ranks, x$rank)
        taxa <- x$taxon[m]
        taxa[startsWith(taxa, "unclassified_")] <- NA
        taxa
}))
colnames(taxid) <- ranks; rownames(taxid) <- getSequences(seqtab.nochim)
```

Create phyloseq object

```
pseq <- phyloseq(otu_table(seqtab.nochim, taxa_are_rows = FALSE),
                 tax_table(taxid))
row.names(metadata) <- sample_names(pseq)
sample_data(pseq) <- metadata
pseq
```

```
phyloseq-class experiment-level object
otu_table()   OTU Table:         [ 30 taxa and 6 samples ]
sample_data() Sample Data:       [ 6 samples by 2 sample variables ]
tax_table()   Taxonomy Table:    [ 30 taxa by 7 taxonomic ranks ]
```

Sequence data is stored as taxa_names. We will store sequences as refseq and create numbered variant names

```
#create DNA object and store sequences
seqs <- DNAStringSet(taxa_names(pseq))
names(seqs) <- taxa_names(pseq)
pseq <- merge_phyloseq(pseq, seqs)
#new variant names
taxa_names(pseq) <- paste0("ASV", seq(ntaxa(pseq)))
#capitalise taxonomic ranks
colnames(tax_table(pseq)) <- c("Kingdom", "Phylum", "Class",
  "Order", "Family", "Genus", "Species")
```

Remove non-bacterial taxa

```
pseq <- subset_taxa(pseq, Kingdom != is.na(Kingdom))
```

Write results to files

Abundance table is transponed and written as tsv file

```
#variant names in rows
ASV_names <- taxa_names(pseq)
#sample names will be columns
ASV_counts <- t(otu_table(pseq))
ASVdf <- (data.frame(ASV_names,ASV_counts))
#write
write_tsv(ASVdf, paste0(exportloc,"asvs.tsv"))
```

Likewise taxonomy table is saved as tsv

```
#variant names in rows
ASV_names <- taxa_names(pseq)
#taxonomy ranks in columns
taxonomy <- (data.frame(ASV_names, tax_table(pseq)))
#write
write_tsv(taxonomy,paste0(exportloc,"taxonomy.tsv"))
```

Variant sequences are saved into fasta file

```
pseq %>% refseq() %>% writeXStringSet(paste0(exportloc,"repseq.fasta"),
                                      append = FALSE, compress = FALSE,
                                      format = "fasta")
```