Microbiome analysis using nanopore reads

Marko Suokas

#### Preprocessing reads

Dorado does not support demultiplexing custom dual indexes located on both the 5’ and 3’ ends. In ligated libraries, these sequences can appear in either orientation. To address this, we use cutadapt for demultiplexing. Index pairs are identified using the linked adapters approach in both forward and reverse orientations. Afterward, a script is applied to reverse complement sequences in the reverse orientation. Finally, the forward and reverse reads are merged for each sample.

#### Extract Forward Reads

Forward reads can be demultiplexed using following command:

cutadapt -e 0 -O 12 -g file:~/scripts/barcodes.fasta --trimmed-only \  
-m 1200 -o "fdemuxed/{name}.fastq.gz" reads.fastq.gz

This command extracts barcodes defined in the barcodes.fasta file and outputs matching reads into individual files within the fdemuxed subdirectory. The minimum read length is set to 1200 bp for 16S amplicons.

#### Extract Reverse Reads

In order to extract reverse reads, the reverse-complemented barcode file is used:

cutadapt -e 0 -O 12 -g file:~/scripts/rev\_barcodes.fasta --trimmed-only \  
-m 1200 -o "rdemuxed/{name}.fastq.gz" reads.fastq.gz

The reads are demultiplexed into a rdemuxed directory.

**Tip:** Parameters -O, -e, -m, and -M can help reduce the chances of mismatched alignments.

#### Reverse Complement Reverse Reads

Next, we use a bash script to process each reverse file and reverse complement them using the following command:

seqkit seq -rp --seq-type DNA -o reverse\_comp.fastq.gz reverse\_out.fastq.gz

#### Merging Forward and Reverse Reads

Subsequent step is to merge forward and reverse reads with the same base name from two directories. Here’s a simple bash command for that:

zcat forward\_out.fastq.gz reverse\_comp.fastq.gz > merged\_reads.fastq.gz

#### Trimming Primers

Finally, cutadapt and bash scripts can be employed to trim forward and reverse PCR primers from the sequence reads.

#### Read quality

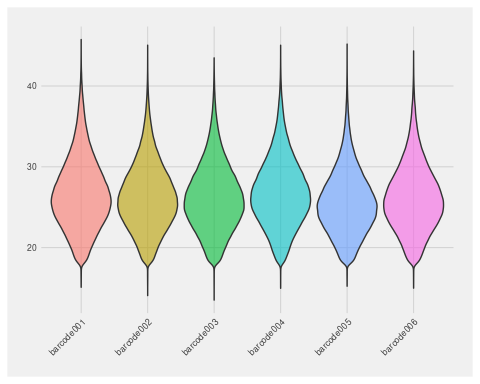
Nanopore sequencing Phred scores should be interpreted with caution, as per-base accuracy cannot be directly determined from the electrical signal. Instead, scores are estimates based on the confidence of the basecalling model rather than a direct probability of sequencing error. As a result, traditional quality filtering approaches may not always be appropriate.

ONT instead calculates an estimated cumulative error rate and converts it into an average quality score per read. The following code computes this metric and generates a violin plot showing the distribution of average sequence quality per read for each sample.

# Define path  
source\_dir <- "data/reads/set1"  
# A function to calculate so called AvgQual  
extract\_nano\_qscores <- function(file) {  
 # Read fastq  
 fq <- readFastq(file)  
 # Extract q-values to a matrix  
 qmat <- as(quality(fq), "matrix")  
 # Convert values to error probabilities  
 error\_probs <- 10^(-qmat/10)  
 # Compute number of expected errors per read  
 total\_errors <- rowSums(error\_probs, na.rm = T)  
 # Length per read  
 read\_lenghts <- rowSums(!is.na(qmat))  
 # Compute "ONT-style" Q-score  
 ont\_qscores <- -10\*log10(total\_errors/read\_lenghts)  
 # Trim file names  
 sampleid <- sub("\\.fastq\\.gz$", "", basename(file))  
   
 # Create a dataframe  
 qdata <- tibble(  
 nanopore\_qscore = ont\_qscores,  
 sampleid = sampleid)  
 return(qdata)  
}  
  
# Create list of files  
files <- list.files(source\_dir, pattern = "\\.fastq\\.gz$",  
 full.names = T)  
# Apply function and use map\_dfr to combine dataframes together  
qscore\_data <- map\_dfr(files, extract\_nano\_qscores)  
# Save result to a rds file  
saveRDS(qscore\_data, "set1/nanopore\_quality.rds")

Violin plot

# Read data  
qdata <- readRDS("set1/nanopore\_quality.rds")  
  
# Divide samples to groups of 12  
sample\_list <- unique(qdata$sampleid)  
sample\_groups <- split(sample\_list, ceiling(seq\_along(sample\_list) / 12))  
  
# Generate a separate plot for each group  
plots <- map(sample\_groups, function(group\_samples) {  
 ggplot(filter(qdata, sampleid %in% group\_samples),  
 aes(x = sampleid, y = nanopore\_qscore, fill = sampleid)) +  
 geom\_violin(scale = "width", alpha = 0.6) +  
 theme\_fivethirtyeight(base\_size=8) +  
 labs(x = "Sample", y = "Nanopore Q-score") +  
 theme(legend.position = "none", axis.text.x = element\_text(angle = 45, hjust = 1))  
})  
  
# Combine all plots into a single vertical layout  
final\_plot <- wrap\_plots(plots, ncol = 1)  
  
# Show the plot  
print(final\_plot)



#### Import sequence data to Qiime 2

QIIME2 import requires manifest.csv that defines names and paths of sequence files.

# Activate qiime environment  
export PATH="/homedir04/msuokas/miniconda3/bin:$PATH"  
source activate qiime-2024.10  
  
# Import sequence data  
qiime tools import \  
 --type 'SampleData[SequencesWithQuality]' \  
 --input-path data/reads/set1/manifest.csv \  
 --output-path data/set1/demux.qza \  
 --input-format SingleEndFastqManifestPhred33

#### Dereplicate sequences

Dereplication will remove sequence redundancy

# Activate qiime environment  
export PATH="/homedir04/msuokas/miniconda3/bin:$PATH"  
source activate qiime-2024.10  
  
# Dereplicate sequences with vsearch plugin  
qiime vsearch dereplicate-sequences \  
 --p-min-seq-length 1200 \  
 --o-dereplicated-table data/set1/derep\_table.qza \  
 --o-dereplicated-sequences data/set1/derep\_sequences.qza \  
 --i-sequences data/set1/demux.qza

#### Pick otus using closed-otu-picking from Silva reference

For low diversity samples closed-otu-picking seems good approach.

# Activate qiime environment  
export PATH="/homedir04/msuokas/miniconda3/bin:$PATH"  
source activate qiime-2024.10  
  
# Closed reference otu clustering command  
qiime vsearch cluster-features-closed-reference \  
 --i-sequences data/set1/derep\_sequences.qza \  
 --i-table data/set1/derep\_table.qza \  
 --i-reference-sequences ~/reference/silva-138-99-seqs.qza \  
 --p-strand plus --p-threads 10 --p-perc-identity 0.97 \  
 --o-clustered-table data/set1/clustered\_table \  
 --o-clustered-sequences data/set1/clustered\_seq.qza \  
 --o-unmatched-sequences data/set1/unmatched.qza

#### Remove rare features

Our sampling is higher than typically and chosen minimum frequency was selected accordingly.

# Activate qiime environment  
export PATH="/homedir04/msuokas/miniconda3/bin:$PATH"  
source activate qiime-2024.10  
  
# QIIME 2 command to filter rare features  
qiime feature-table filter-features \  
--i-table data/set1/clustered\_table.qza \  
--p-min-frequency 10 \  
--o-filtered-table data/set1/n10\_table.qza  
  
# QIIME 2 command to filter sequence file  
qiime feature-table filter-seqs \  
 --i-data data/set1/clustered\_seq.qza \  
 --i-table data/set1/n10\_table.qza \  
 --o-filtered-data data/set1/n10\_seq.qza

#### Detect and filter chimeric features

Detecting chimeric features after removal of rare features will save computing resources.

# Activate qiime environment  
export PATH="/homedir04/msuokas/miniconda3/bin:$PATH"  
source activate qiime-2024.10  
  
# Detect chimeras  
qiime vsearch uchime-denovo \  
--i-sequences data/set1/n10\_seq.qza \  
--i-table data/set1/n10\_table.qza \  
--o-chimeras data/set1/chimeras.qza \  
--o-stats data/set1/uchime-stats.qza \  
--o-nonchimeras data/set1/nonchimeras.qza  
  
# Keep nonchimeric features in the table  
qiime feature-table filter-features \  
--i-table data/set1/n10\_table.qza \  
--m-metadata-file data/set1/nonchimeras.qza \  
--o-filtered-table data/set1/otu\_table.qza  
  
# Keep nonchimeric sequences  
qiime feature-table filter-seqs \  
 --i-data data/set1/n10\_seq.qza \  
 --i-table data/set1/otu\_table.qza \  
 --o-filtered-data data/set1/otu\_seq.qza

#### Import qiime files and metadata

We will import feature table along with representative sequences and metadata to a TSE object.

#sequence file  
tse <- importQIIME2(featureTableFile = "data/set1/otu\_table.qza",  
 refSeqFile = "data/set1/otu\_seq.qza")  
tse <- tse[, sort(colnames(tse))]  
#add metadata  
metadata <- data.frame(read\_tsv("data/set1\_meta.tsv",  
 show\_col\_types = F))  
metadata <- column\_to\_rownames(metadata, "Sampleid")  
colData(tse) <- DataFrame(metadata)  
tse

class: TreeSummarizedExperiment   
dim: 95 6   
metadata(0):  
assays(1): counts  
rownames(95): CP016294.613081.614615 JF312983.1.1520 ...  
 FR682931.1.1497 AB294555.1.1541  
rowData names(0):  
colnames(6): barcode001 barcode002 ... barcode005 barcode006  
colData names(1): Name  
reducedDimNames(0):  
mainExpName: NULL  
altExpNames(0):  
rowLinks: NULL  
rowTree: NULL  
colLinks: NULL  
colTree: NULL  
referenceSeq: a DNAStringSet (95 sequences)

#### Create taxonomy table from Silva

As we have used Silva to our otu picking, we can fetch taxonomic information using Silva sequnence identifiers.

# Routine to separate taxonomic ranks  
silva\_tax <- read\_tsv("~/reference/silva\_taxonomy.tsv",  
 show\_col\_types = F)  
rows <- data.frame(FeatureID = rownames(tse))  
taxonomy <- dplyr::left\_join(rows, silva\_tax, by = "FeatureID")  
taxonomy <- taxonomy %>%  
 separate(Taxon,   
 into = c("Kingdom", "Phylum", "Class", "Order", "Family", "Genus", "Species"),   
 sep = "; ",   
 fill = "right")  
taxonomy <- taxonomy %>%  
 mutate(across(everything(), ~ sub("^[a-z]\_\_", "", .)))  
taxonomy <- column\_to\_rownames(taxonomy, "FeatureID")

Species-level information is filtered to remove ambiguous matches.

taxonomy <- taxonomy %>%  
 mutate(Species = if\_else(str\_detect(Species, "unknown|uncultured|metagenome"),  
 NA\_character\_, Species))

Add taxonomy to rowData

# Add taxonomy  
rownames(taxonomy) <- NULL  
rowData(tse) <- DataFrame(taxonomy)  
# Check object  
tse

class: TreeSummarizedExperiment   
dim: 95 6   
metadata(0):  
assays(1): counts  
rownames(95): CP016294.613081.614615 JF312983.1.1520 ...  
 FR682931.1.1497 AB294555.1.1541  
rowData names(7): Kingdom Phylum ... Genus Species  
colnames(6): barcode001 barcode002 ... barcode005 barcode006  
colData names(1): Name  
reducedDimNames(0):  
mainExpName: NULL  
altExpNames(0):  
rowLinks: NULL  
rowTree: NULL  
colLinks: NULL  
colTree: NULL  
referenceSeq: a DNAStringSet (95 sequences)

#### Write data to files

Write RDS. It can be reloaded to R

saveRDS(tse, "set1/tse.rds")

Write abundance table.

#FeatureID will be rowname  
abd <- data.frame(FeatureID = rownames(tse),assays(tse)$counts)  
#Write  
write\_tsv(abd, "set1/otu\_table.tsv")

Write taxonomy table.

#FeatureID will be rowname  
taxt <- data.frame(FeatureID = rownames(tse), rowData(tse))  
#Write  
write\_tsv(taxt, "set1/taxonomy.tsv")

Write sequences to fasta file.

writeXStringSet(referenceSeq(tse), "set1/repseq.fasta",  
 append = F, compress = F,  
 format = "fasta")

Write metadata file.

metadf <- data.frame(colData(tse)) %>% rownames\_to\_column(var="Sampleid")  
#write  
write\_tsv(metadf, "set1/metadata.tsv")

#### Microbe data analysis

Agglomerate taxonomy to genus-level and count relative abundance

# Agglomerate  
tse <- agglomerateByRank(tse, rank = "Genus",  
 onRankOnly = T, na.rm = F)  
# Relabundance  
tse <- transformAssay(tse, assay.type = "counts", method = "relabundance")

Pick most abundant features

#top features  
top10 <- getTop(tse, top = 7, method = "mean",  
 assay.type = "relabundance")  
#create and filter table  
table <- data.frame(assays(tse)$relabundance)  
table <- table %>% rownames\_to\_column(var = "Genus") %>%   
 filter(Genus %in% top10)

Composition table

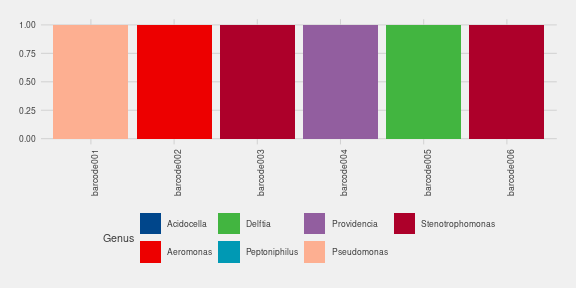
kable(table, digits=5) %>%   
 kable\_styling(latex\_options = c("HOLD\_position", "striped"),  
 font\_size = 8) %>%  
 row\_spec(0, background = "teal", color = "white")

| Genus | barcode001 | barcode002 | barcode003 | barcode004 | barcode005 | barcode006 |
| --- | --- | --- | --- | --- | --- | --- |
| Acidocella | 0.00002 | 0.00000 | 0.00000 | 0.00000 | 0.00001 | 0.00000 |
| Aeromonas | 0.00001 | 0.99997 | 0.00001 | 0.00000 | 0.00000 | 0.00002 |
| Delftia | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.99999 | 0.00001 |
| Peptoniphilus | 0.00000 | 0.00000 | 0.00000 | 0.00003 | 0.00000 | 0.00000 |
| Providencia | 0.00001 | 0.00000 | 0.00000 | 0.99996 | 0.00001 | 0.00000 |
| Pseudomonas | 0.99995 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| Stenotrophomonas | 0.00000 | 0.00003 | 0.99999 | 0.00000 | 0.00000 | 0.99997 |

#### Composition plot

Convert data first to long table format and then plot.

# Long dataframe  
df\_long <- table %>% pivot\_longer(cols = starts\_with("barcode"),  
names\_to = "Sample", values\_to = "Abundance")  
  
# Plot stacked barplot  
ggplot(df\_long, aes(x = Sample, y = Abundance, fill = Genus)) +  
geom\_bar(stat = "identity") + theme\_fivethirtyeight(base\_size = 8) + scale\_fill\_lancet() + theme(axis.text.x =  
element\_text(angle = 90))



#### Summary of closed reference otu picking strategy

Count first number of sequences in raw files.

# List all compressed fastq files in the folder  
fastq\_files <- list.files("data/reads/set1", pattern = "\\.fastq\\.gz$", full.names = T)  
  
# Function to count sequences in a compressed FASTQ file  
count\_sequences\_in\_fastq <- function(file) {  
 fq <- readFastq(file)  
 return(length(fq))  
}  
# Apply the function to each file and store the counts  
sequence\_counts <- sapply(fastq\_files, count\_sequences\_in\_fastq)  
# Print the result  
raw\_data <- data.frame(File = fastq\_files, Sequences = sequence\_counts)

Create summary table

summary <- data.frame(Sample = colnames(tse), Raw\_Counts = raw\_data$Sequences, Counts = colSums(assays(tse)$counts))  
summary <- summary %>% mutate(Percentage = Counts/Raw\_Counts)  
rownames(summary) <- NULL  
kable(summary, digits=2) %>%   
 kable\_styling(latex\_options = c("HOLD\_position", "striped"),  
 font\_size = 12) %>%  
 row\_spec(0, background = "teal", color = "white")

| Sample | Raw\_Counts | Counts | Percentage |
| --- | --- | --- | --- |
| barcode001 | 391434 | 308345 | 0.79 |
| barcode002 | 394282 | 362506 | 0.92 |
| barcode003 | 401407 | 400226 | 1.00 |
| barcode004 | 370675 | 358410 | 0.97 |
| barcode005 | 388126 | 382881 | 0.99 |
| barcode006 | 371125 | 359805 | 0.97 |

#### Results

Results show that using SUP basecalling with additional quality filtering produce sequence accuracy that is enough for OTU picking stategies via sequence clustering. Most of nanopore studies use mapping strategies that might not always recognize all biological features.

The samples were purified cultures, and a closed-reference OTU-picking strategy was chosen accordingly. Our results confirm that all samples were effectively pure cultures. In fact, exceptionally low noise level at genus level results suggest that predicted 0.5-1 bp errors / 100 bp sequence is realistic.

Additionally, proportion of matched sequences against database was high across all samples suggesting that strategy is a viable option when analysing microbial communities with nanopore sequencer.