**Workflow, commands and scripts**

A. Make sure that each sample has its own fastq file. We have to demultiplex if they have been merged in one file. Use this to demultiplex in the Terminal (<https://astrobiomike.github.io/amplicon/demultiplexing>):

*column -t 122115BHits7F-mapping.txt| head*

*awk -v OFS="\t" ' NR > 1 {print $2, $1"\_R1.fq", $1"\_R2.fq"} ' 122115BHits7F-mapping.txt > sabre\_formatted\_barcode\_file.txt*

*sabre pe -f SAM1-23\_S3\_L001\_R1\_001.fq -r SAM1-23\_S3\_L001\_R2\_001.fq -b sabre\_formatted\_barcode\_file.txt -u no\_bc\_match\_R1.fq -w no\_bc\_match\_R2.fq*

\*Highlighted in yellow are names that would need to be changed, depending on the name of the files.

B. Use Cutadapt to remove primers

#Use Python 3.7.10

*cutadapt --version # 2.3*

*for sample in $(cat samples)*

*do*

*echo "On sample: $sample"*

*cutadapt -a ^GTGARTCATCGAATCTTTG...GCATATCAATAAGCGGAGGA -A ^TCCTCCGCTTATTGATATGC...CAAAGATTCGATGAYTCAC -m 215 -M 285 --discard-untrimmed -o ${sample}\_R1\_trimmed.fq.gz -p ${sample}\_R2\_trimmed.fq.gz ${sample}\_R1.fq ${sample}\_R2.fq >> cutadapt\_primer\_trimming\_stats.txt 2>&1*

*done*

Now the \*trimmed.fq.gz files can be used in Dada2 in RStudio

Unzip files

C. Dada2 in R Studio

#### Workflow modified from https://astrobiomike.github.io/amplicon/dada2\_workflow\_ex

#and https://benjjneb.github.io/dada2/tutorial.html ####

#start Dada

library(dada2)

####SETTING THE WORK ENVIRONMENT#####

###########################################

setwd("~/{path}")

##Define the path variable ##

path <-("~/{path}/")

list.files(path)

# Forward and reverse fastq filenames have format: SAMPLENAME\_R1.fq and SAMPLENAME\_R2.fq

fnFs <- sort(list.files(path, pattern="\_R1.fq", full.names = TRUE))

fnRs <- sort(list.files(path, pattern="\_R2.fq", full.names = TRUE))

# Extract sample names, assuming filenames have format: SAMPLENAME\_XXX.fq

sample.names <- sapply(strsplit(basename(fnFs), "\_"), `[`, 1)

##### Inspect read quality profiles #####

# We start by visualizing the quality profiles of the forward reads: #

plotQualityProfile(fnFs[1:2])

# See where the Quality score drops below 20. In the case of this data, it appears as if the quality drops at about position 270 bp; so, trim the last 30 bp #

# We then visualize the quality profiles of the reverse reads: #

plotQualityProfile(fnRs[1:2])

# See where the Quality score drops below 20. In the case of this data, it appears as if after position 210 bp the quality drops # trim the last 90 #

##############################

###### Filter and trim ########

##############################

# Assign the filenames for the filtered fastq.gz files. #

# Place filtered files in filtered/ subdirectory

filtFs <- file.path(path, "filtered", paste0(sample.names, "\_F\_filt.fastq.gz"))

filtRs <- file.path(path, "filtered", paste0(sample.names, "\_R\_filt.fastq.gz"))

names(filtFs) <- sample.names

names(filtRs) <- sample.names

# We’ll use standard filtering parameters:

# maxN=0 (DADA2 requires no Ns),

# truncQ=2, rm.phix=TRUE and maxEE=2.

# The maxEE parameter sets the maximum number of “expected errors” allowed

# in a read, which is a better filter than simply averaging quality scores.

# trunclean, we are using (270,210)

out <- filterAndTrim(fnFs, filtFs, fnRs, filtRs, truncLen=c(270,210),

maxN=0, maxEE=c(2,2), truncQ=2, rm.phix=TRUE,

compress=TRUE, multithread=TRUE) # On Windows set multithread=FALSE

head(out)

#To save a file with the results (out file), use this:

write.table(out, "Output.tsv", sep="\t", quote=F, col.names=NA)

##### Learn the Error Rates ######

#############################

errF <- learnErrors(filtFs, multithread=TRUE)

# This is for the Forward reads #

errR <- learnErrors(filtRs, multithread=TRUE)

# This is for the Reverse reads #

plotErrors(errF, nominalQ=TRUE)

# to plot and visualize the errors of the Forward reads#

# # The developers have incorporated a plotting function to visualize

# how well the estimated error rates match up with the observed:#

###But generally speaking, you want the observed (black dots) to track

# well with the estimated (black line).####

plotErrors(errR, nominalQ=TRUE)

# to plot and visualize the errors of the Reverse reads#

###################################

### Sample Inference #############

### This is where we will get clean ASVs, after learning error rates #

##### and using filtered sequences #

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

# For the forward reads #

dadaRs <- dada(filtRs, err=errR, multithread=TRUE)

# For the reverse reads #

### Inspecting the returned dada-class object: ###

## dada-class: object describing DADA2 denoising results

## xxx sequence variants were inferred from xxx input unique sequences.

## Key parameters: OMEGA\_A = 1e-40, OMEGA\_C = 1e-40, BAND\_SIZE = 16

dadaFs[[1]] # is for forward

dadaRs[[1]] # is for reverse

#####################################

##### Merge paired reads ###########

######################################

# We now merge the forward and reverse reads #

# to obtain the full denoised sequences.#

mergers <- mergePairs(dadaFs, filtFs, dadaRs, filtRs, verbose=TRUE)

# Inspect the merger data.frame from the first sample

head(mergers[[1]])

# The 'mergers' object is a list of data.frames from each sample.

# Each data.frame contains the merged $sequence, its $abundance,

# and the indices of the $forward and $reverse sequence variants that were merged.

# Paired reads that did not exactly overlap were removed by mergePairs,

# further reducing spurious output. #

################################################

####### Construct sequence table ###############

################################################

## We can now construct an amplicon sequence variant table (ASV) table####

seqtab <- makeSequenceTable(mergers)

dim(seqtab)

## The first number is the number of samples and the second is the ASVs

# Inspect distribution of sequence lengths

table(nchar(getSequences(seqtab)))

## The sequence table is a matrix with rows corresponding to (and named by)

# the samples, and columns corresponding to (and named by) the sequence variants.

# This table contains 6944 ASVs, and the lengths of our merged sequences all fall

# within the expected range for this ITS amplicon (~250). ##

############################################

############ Remove chimeras ##############

##########################################

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

dim(seqtab.nochim)

# the resulting numbers are 'number of samples' and 'number of ASVs" left that

# are not chimeras ##

###################################################

############ Track reads through the pipeline ######

####################################################

## As a final check of our progress, we’ll look at the number of reads that

# made it through each step in the pipeline: ##

getN <- function(x) sum(getUniques(x))

track <- cbind(out, sapply(dadaFs, getN), sapply(dadaRs, getN), sapply(mergers, getN), rowSums(seqtab.nochim))

# If processing a single sample, remove the sapply calls:

# e.g. replace sapply(dadaFs, getN) with getN(dadaFs) #

colnames(track) <- c("input", "filtered", "denoisedF", "denoisedR", "merged", "nonchim")

rownames(track) <- sample.names

head(track)

#This next one is to save a table with all the sample information: unput, filtered,

# denoisedF, denoisedR, merged, and nonchimeric #

write.table(track, "track.tsv", sep="\t", quote=F, col.names=NA)

seqtabFinal <- seqtab.nochim

library(DECIPHER)

# Check abundance of reads on every sample

library(dplyr)

library(tibble)

transformed <- seqtabFinal %>%

t() %>%

as.data.frame() %>%

tibble::rownames\_to\_column("sequence")

### transformed is a data frame, the first column is the inferred sequence and

# the other columns are the abundance in each sample ##

################################################

############ Taxonomy assignment using DECIPHER #############

################################################

# Trained classifiers are available from http://DECIPHER.codes/Downloads.html.

# Download the UNITED (modified) file to follow along.

# UNITE\_v2020\_February2020.RData ###

##

dna <- DNAStringSet(getSequences(seqtabFinal)) # Create a DNAStringSet from the ASVs

#Load the trained set UNITE\_v2020.RData#

load("~/{path}/UNITE\_v2020.RData")

ids <- IdTaxa(dna, trainingSet, strand="both", processors=NULL, verbose=TRUE) # use all processors

ranks <- c("kingdom", "phylum", "class", "order", "family", "genus", "species") # ranks of interest

# Convert the output object of class "Taxa" 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[startsWith(taxa, "unidentified\_")] <- NA

taxa[startsWith(taxa, "unidentified")] <- NA

taxa[startsWith(taxa, "Fungi\_sp")] <- NA

taxa[startsWith(taxa, "Ascomycota\_sp")] <- NA

taxa

}))

colnames(taxid) <- ranks; rownames(taxid) <- getSequences(seqtabFinal)

# Give sequence headers more manageable names (ASV\_1, ASV\_2...)

asv\_seqs <- colnames(seqtabFinal)

asv\_headers <- vector(dim(seqtabFinal)[2], mode="character")

for (i in 1:dim(seqtabFinal)[2]) {

asv\_headers[i] <- paste(">ASV", i, sep="\_")

}

# making and writing out a fasta of our final ASV seqs:

asv\_fasta <- c(rbind(asv\_headers, asv\_seqs))

write(asv\_fasta, "ASVs.fa")

#Write: is for saving a fasta file in the working folder.

# count table:

asv\_tab <- t(seqtabFinal)

row.names(asv\_tab) <- sub(">", "", asv\_headers)

write.table(asv\_tab, "ASVs\_counts.tsv", sep="\t", quote=F, col.names=NA)

# taxonomy table:

# creating table with taxonomy labels and ASVs

row.names(taxa) <- sub(">", "", asv\_headers)

write.table(taxa, "ASV\_Taxonomy.tsv", sep="\t", quote=F, col.names=NA)