Data processing for BB2560 - report

Sumant Salphale

2022-03-17

Part I Assessing Quality of Data

We begin by loading the required packages for the lab, then we load the data files we are working with and we look at the quality and then filter the data before we begin any analysis.

```
library(dada2)
library(edgeR)
library(pheatmap)
library(vegan)
```

```
list.files()
```

```
sample_info = read.delim("BB2560_lab2_Feb2022_sample_info.txt")
ngi_sample_id = sample_info[,1]
sample_name = sample_info[,2]
sample_type  = sample_info[,3]
fnFs = list.files(pattern="_R1_001.fastq")
fnRs = list.files(pattern="_R2_001.fastq")

fnFs = list.files(pattern="_R1_001.fastq")
fnRs = list.files(pattern="_R2_001.fastq")
```

```
plotQualityProfile(fnFs[1:4])
plotQualityProfile(fnRs[1:4])
```

```
filtFs = file.path("filtered", paste0(ngi_sample_id,"_F_filt.fastq.gz"))
filtRs = file.path("filtered", paste0(ngi_sample_id,"_R_filt.fastq.gz"))
```

```
out = filterAndTrim(
  fnFs, filtFs, fnRs, filtRs,
  truncLen=c(180,180), trimLeft=c(0,0),
  maxN=0, maxEE=c(2,2), truncQ=2, rm.phix=TRUE,
  compress=TRUE, multithread=FALSE
)
```

```
plotQualityProfile(filtFs[1:4])
plotQualityProfile(filtRs[1:4])
```

Part II ASV table Construction

Now we can start working with the data. We calculate error rates, dereplicate the data and infer true sequence variants. Finally we construct an ASV table that we can continue working with.

```
# Calculate error rates
errF = learnErrors(filtFs, multithread=TRUE)
errR = learnErrors(filtRs, multithread=TRUE)
```

```
# Run dereplication
derepFs = derepFastq(filtFs, verbose=FALSE)
derepRs = derepFastq(filtRs, verbose=FALSE)

# Name the derep-class objects by the sample names:
names(derepFs) = sample_name
names(derepRs) = sample_name

# Check the amount of reads for one sample
derepFs$`B6SS`
derepRs$`B6SS`
```

```
# Infer the true sequence variants from the unique sequences
dadaFs = dada(derepFs, err=errF, multithread=TRUE, verbose = FALSE)
dadaRs = dada(derepRs, err=errR, multithread=TRUE, verbose = FALSE)

# Check the amount of denoised reads for one sample
dadaFs$`B6SS`
dadaRs$`B6SS`
```

```
# From the denoised data we construct an amplicon sequence variant table (ASV).
mergers = mergePairs(dadaFs, derepFs, dadaRs, derepRs, verbose=FALSE)
seqtab = makeSequenceTable(mergers)
#Check total number of ASVs present in the table
length(seqtab)
```

```
# Remove any present chimeras from the ASV table.
seqtab = removeBimeraDenovo(
  seqtab, method="consensus", multithread=TRUE, verbose=FALSE)

seqtab = t(seqtab)
#Check how many ASVs remain in the table after removal of chimeras
length(seqtab)
```

```
# Check the column names
colnames(seqtab)

# Check the row names
rownames(seqtab)

# Copy the ASV sequences into a new vector
asv = rownames(seqtab)
```

```r
# Rename the rows
rownames(seqtab) = paste("ASV", 1:length(asv), sep = "")

# Check the new row names
rownames(seqtab)
```

```r
# Determine the number of chimeras in sourdough and kombucha samples

GM = which(sample_type== "Goat milk")
GCF = which(sample_type== "Goat cheese France")
GCS = which(sample_type == "Goat cheese Spain")
goatmilk = sample_info$Course_ID[GM]
goatcheesef = sample_info$Course_ID[GCF]
goatcheeses = sample_info$Course_ID[GCS]
RM = which(sample_type == "Cow milk raw")
BC = which(sample_type == "Blue cheese")
cowmilk = sample_info$Course_ID[RM]
bluecheese = sample_info$Course_ID[BC]
OM = which(sample_type=="Oat milk")
OY = which(sample_type=="Oat yoghurt")
oatmilk = sample_info$Course_ID[OM]
oatyoghurt = sample_info$Course_ID[OY]

for (i in goatmilk) {
  p = length(which(seqtab[,i]>0))
  cat("Number of uniques ASVs in goat milk sample",i, "are:", p, "\n")
}
for (i in goatcheesef) {
  p = length(which(seqtab[,i]>0))
  cat("Number of unique ASVs in goat cheese france sample",i, "are:", p, "\n")
}
for (i in goatcheeses) {
  p = length(which(seqtab[,i]>0))
  cat("Number of unique ASVs in goat cheese spain sample",i, "are:", p, "\n")
}
for (i in cowmilk) {
  p = length(which(seqtab[,i]>0))
  cat("Number of unique ASVs in cow milk sample",i, "are:", p, "\n")
}
for (i in bluecheese) {
  p = length(which(seqtab[,i]>0))
  cat("Number of unique ASVs in blue cheese sample",i, "are:", p, "\n")
}
for (i in oatmilk) {
  p = length(which(seqtab[,i]>0))
  cat("Number of unique ASVs in oat milk sample",i, "are:", p, "\n")
}
for (i in oatyoghurt) {
  p = length(which(seqtab[,i]>0))
  cat("Number of unique ASVs in oat yoghurt sample",i, "are:", p, "\n")
}
```

```r
# use the function `assignTaxonomy` in the DADA2 package to get a taxonomic label of each ASV.
set.seed(123)
taxa = assignTaxonomy(
  asv, "silva_nr99_v138.1_train_set.fa.gz",
  multithread=TRUE,
  taxLevels = c("Domain", "Phylum", "Class", "Order", "Family", "Genus")
)

taxa <- addSpecies(taxa, "silva_species_assignment_v138.1.fa.gz", allowMultiple = FALSE)
#allowMultiple (Optional). Default FALSE. Defines the behavior when exact matches to multiple (differen

# we name the rows of taxa as the rows of seqtab, ie with ASV ids
rownames(taxa) = rownames(seqtab)
```

```r
# check the number of ASVs at different taxonomic levels
for (i in (1:7)) {
  cat("Number of ASVs at", colnames(taxa)[[i]], "level are", sum(!is.na(taxa[,i])), "\n")
}
```

Part III Data Analysis (Taxonomy, Alpha and Beta diversity)

Now we get to the actual analysis of ur data

```r
# Normalise the data in the seqtab dataframe
norm_seqtab = seqtab
for (i in 1:ncol(seqtab)) {
  norm_seqtab[,i] = seqtab[,i]/sum(seqtab[,i])
}
```

```r
# Summarise the ASV count data at broader taxonomic levels, such as at the
# phylum level. Make two count matrices, one with raw counts and one with
# normalised counts, per taxonomic level (from domain to genus).

clade_counts = list()
norm_clade_counts = list()

for (i in 1:ncol(taxa)) {
  matr = norm_matr = NULL
  clade = unique(taxa[,i])
  clade = clade[!is.na(clade)]
  for (j in 1:length(clade)) {
    ix = which(clade[j]==taxa[,i])
    if (length(ix) > 1) {
      matr = rbind(matr, apply(seqtab[ix,], 2, sum, na.rm=TRUE))
      norm_matr = rbind(norm_matr, apply(norm_seqtab[ix,], 2, sum, na.rm=TRUE))
    } else {
      matr = rbind(matr, seqtab[ix,])
      norm_matr = rbind(norm_matr, norm_seqtab[ix,])
    }
  }
  rownames(matr) = rownames(norm_matr) = clade
  colnames(matr) = colnames(norm_matr) = sample_name
  clade_counts[[i]] = matr
  norm_clade_counts[[i]] = norm_matr
}
```

```
# Let's check the counts of all phyla in the first sample:
clade_counts[[2]][,1]

# Or the counts of the third genus in samples 1-10:
clade_counts[[6]][3,1:10]
```

```
# Check the number of different samples at different taxonomic levels that are present in the sample an

tax_level = 6 # choose taxonomic level 1-7

for (i in GM) {
  p = length(which(clade_counts[[tax_level]][,i]>0))
  q = sum(clade_counts[[tax_level]][,i])
  r = sample_info$Course_ID[i]
  cat("Number of sample present at level", colnames(taxa)[[tax_level]] ,"in goat milk sample", r, "are"
}

for (i in GCF) {
  p = length(which(clade_counts[[tax_level]][,i]>0))
  q = sum(clade_counts[[tax_level]][,i])
  r = sample_info$Course_ID[i]
  cat("Number of sample present at level", colnames(taxa)[[tax_level]] ,"in goat cheese France sample",
}
for (i in GCS) {
  p = length(which(clade_counts[[tax_level]][,i]>0))
  q = sum(clade_counts[[tax_level]][,i])
  r = sample_info$Course_ID[i]
  cat("Number of sample present at level", colnames(taxa)[[tax_level]] ,"in goat cheese Spain sample", r
}
for (i in RM) {
  p = length(which(clade_counts[[tax_level]][,i]>0))
  q = sum(clade_counts[[tax_level]][,i])
  r = sample_info$Course_ID[i]
  cat("Number of sample present at level", colnames(taxa)[[tax_level]] ,"in Cow milk Raw sample", r, "a
}
for (i in BC) {
  p = length(which(clade_counts[[tax_level]][,i]>0))
  q = sum(clade_counts[[tax_level]][,i])
  r = sample_info$Course_ID[i]
  cat("Number of sample present at level", colnames(taxa)[[tax_level]] ,"in Blue cheese sample", r, "ar
}
for (i in OM) {
  p = length(which(clade_counts[[tax_level]][,i]>0))
  q = sum(clade_counts[[tax_level]][,i])
  r = sample_info$Course_ID[i]
  cat("Number of sample present at level", colnames(taxa)[[tax_level]] ,"in Oat milk sample", r, "are",
}
for (i in OY) {
  p = length(which(clade_counts[[tax_level]][,i]>0))
  q = sum(clade_counts[[tax_level]][,i])
  r = sample_info$Course_ID[i]
  cat("Number of sample present at level", colnames(taxa)[[tax_level]] ,"in Oat Yoghurt sample", r, "are
}
```

Having the clade count tables we can easily make illustrative barplots of the taxonomic composition:

```
#Grouping goat milk and cheese samples under 'goat'
GM = which(sample_type=="Goat milk")
GCF = which(sample_type=="Goat cheese France")
GCS = which(sample_type=="Goat cheese Spain")
goat = c(GM, GCF, GCS)

#Grouping cow milk and blue cheese under 'cow'
RM = which(sample_type=="Cow milk raw")
BC = which(sample_type=="Blue cheese")
cow = c(RM, BC)

#Grouping oat milk and yoghurt under 'oat'
OM = which(sample_type=="Oat milk")
OY = which(sample_type=="Oat yoghurt")
oat = c(OM, OY)

# set what taxonomic level to plot (1 - 6, corresponding to domain - genus)
tax_level = 4
sample = c(goat,cow,oat)

# to select those clades with a relative abundance over a threshold (here 0.01)
ok = which(apply(norm_clade_counts[[tax_level]], 1, mean) > 0.01)

# save the plot to a folder
setwd("C:/users/ssalp/OneDrive/Documents/Advanced Microbiology and Metagenomics/Images for Report")
png(paste(format("taxplot_selection"), "png", sep="."), width=12, height=10,
units="in", res=300)

# to make a color palette
mycols = colorRampPalette(c("#a6cee3",
                            "#1f78b4",
                            "#b2df8a",
                            "#33a02c",
                            "#fb9a99",
                            "#e31a1c",
                            "#fdbf6f",
                            "#ff7f00",
                            "#cab2d6",
                            "#6a3d9a",
                            "#ffff99",
                            "#b15928"))

# define the plotting area
par(mfrow=c(1,1), mar=c(9,3,0,10), xpd = TRUE)
```

```
# make the barplot
barplot(
  norm_clade_counts[[tax_level]][ok,sample_info$Course_ID[sample]],
  col = mycols(length(ok)),
  las = 2,
  names.arg = paste(sample_type[sample], "\n",sample_info$Course_ID[sample]),
  #horiz=T
  cex.names = 0.8
)

# add a color legend
legend(
  "bottomleft", bty = "n", pch = 19,
  col = mycols(length(ok))[1:length(ok)],
  cex = 1, inset = c(1,0),
  legend = rownames(clade_counts[[tax_level]])[ok]
)
dev.off()
```

```
Activia_yoghurt = which(sample_type=="Activia yoghurt")
Blue_cheese = which(sample_type=="Blue cheese")
Cow_milk_raw = which(sample_type=="Cow milk raw")
Goat_cheese_France = which(sample_type=="Goat cheese France")
Goat_cheese_Spain = which(sample_type=="Goat cheese Spain")
Goat_milk = which(sample_type=="Goat milk")
kimchi_shopbought = which(sample_type=="kimchi shopbought")
Kombucha_homemade = which(sample_type=="Kombucha homemade")
Kombucha_shopbought = which(sample_type=="Kombucha shopbought")
Nypon_proviva = which(sample_type=="Nypon proviva")
Oat_milk = which(sample_type=="Oat milk")
Oat_yoghurt = which(sample_type=="Oat yoghurt")
Sourdough_active = which(sample_type=="Sourdough active")
Sourdough_dry = which(sample_type=="Sourdough dry")
Soy_yoghurt = which(sample_type=="Soy yoghurt")

all = c(Activia_yoghurt, Blue_cheese, Cow_milk_raw, Goat_cheese_France, Goat_cheese_Spain, Goat_milk, k
```

```
# set what taxonomic level to plot (1 - 6, corresponding to domain - genus)
tax_level = 4

# to select those clades with a relative abundance over a threshold (here 0.01)
ok = which(apply(norm_clade_counts[[tax_level]], 1, mean) > 0.01)

# save the plot to a folder
setwd("C:/users/ssalp/OneDrive/Documents/Advanced Microbiology and Metagenomics/Images for Report")
png(paste(format("tax_plot_all"), "png", sep="."), width=12, height=10,
units="in", res=300) #heatmap1.png"

# to make a color palette
mycols = colorRampPalette(c("#a6cee3",
                            "#1f78b4",
                            "#b2df8a",
                            "#33a02c",
                            "#fb9a99",
                            "#e31a1c",
                            "#fdbf6f",
                            "#ff7f00",
                            "#cab2d6",
                            "#6a3d9a",
                            "#ffff99",
                            "#b15928"))

# define the plotting area
par(mfrow=c(1,1), mar=c(9,3,2,10), xpd = TRUE)

# make the barplot
barplot(
  norm_clade_counts[[tax_level]][ok,all],
  col = mycols(length(ok)),
  las = 2,
  names.arg = paste(sample_type[all], sample_info$Course_ID[all]),
  #horiz=T
  cex.names = 0.5
)

# add a color legend
legend(
  "bottomleft", bty = "n", pch = 19,
  col = mycols(length(ok))[1:length(ok)],
  cex = 1, inset = c(1,0),
  legend = rownames(clade_counts[[tax_level]])[ok]
)
dev.off()
```

```
# Calculate Shannon diversity for every sample and put in a vector named shannon
shannon = diversity(seqtab, MARGIN = 2)

# save the plot to a folder
setwd("C:/users/ssalp/OneDrive/Documents/Advanced Microbiology and Metagenomics/Images for Report")
png(paste(format("shannon_all"), "png", sep="."), width=12, height=10,
units="in", res=300) #heatmap1.png"

# We can make a bargraph of the Shannon diversities.
# - We order the samples using the "all" group
# define the plotting area
par(mfrow=c(1,1), mar=c(7,3,2,2), xpd = TRUE)
barplot(
  shannon[all], las = 2,
  names.arg = paste(sample_type[all], sample_name[all]),
  cex.names = 0.5
)
dev.off()
```

```
# or summarise them in boxplots, one per sample group/treatment

# save the plot to a folder
```

```
setwd("C:/users/ssalp/OneDrive/Documents/Advanced Microbiology and Metagenomics/Images for Report")
png(paste(format("shannon_box"), "png", sep="."), width=12, height=10,
units="in", res=300) #heatmap1.png"

# define the plotting area
par(mfrow=c(1,1), mar=c(3,3,3,3), xpd = TRUE)
boxplot(
  shannon[RM], shannon[BC],
  shannon[GM], shannon[GCF], shannon[GCS],
  shannon[OM], shannon[OY],
  names = c(
    "Cow milk raw", "blue cheese",
    "goat milk", "goat cheese france", "goat cheese spain",
    "oat milk", "oat yoghurt"
),las = 1
)
dev.off()
```

```
# Run a Wilcox test on the samples
wilcox.test(shannon[RM], shannon[BC])
wilcox.test(shannon[GM], shannon[GCF])
wilcox.test(shannon[GM], shannon[GCS])
wilcox.test(shannon[OM], shannon[OY])
```

```
# Run the paired test
wilcox.test(
  shannon[tail(GM,3)],
  shannon[tail(GCF,3)],
  paired=TRUE
)
```

```
bray_dist = as.matrix(vegdist(t(norm_seqtab), method = "bray"))

# Visualise the distance matrix as a heatmap

# save the plot to a folder
setwd("C:/users/ssalp/OneDrive/Documents/Advanced Microbiology and Metagenomics/Images for Report")
png(paste(format("heatmap1"), "png", sep="."), width=12, height=10,
units="in", res=300) #heatmap1.png"

# Visualise the distance matrix as a heatmap
pheatmap(
  bray_dist,
  cluster_rows = FALSE, cluster_cols = FALSE,
  labels_row = paste(sample_name, sample_type),
  labels_col = paste(sample_name, sample_type),
)
dev.off()
```

```
bray_dist = as.matrix(vegdist(t(norm_seqtab), method = "bray"))

# save the plot to a folder
setwd("C:/users/ssalp/OneDrive/Documents/Advanced Microbiology and Metagenomics/Images for Report")
```

```r
png(paste(format("heatmap2"), "png", sep="."), width=12, height=10,
units="in", res=300) #heatmap1.png"

# Visualise the distance matrix as a clustered heatmap
pheatmap(
  bray_dist,
  clustering_distance_rows = as.dist(bray_dist),
  clustering_distance_cols = as.dist(bray_dist),
  labels_row = paste(sample_name, sample_type),
  labels_col = paste(sample_name, sample_type),
)
dev.off()
```

```r
samples = c(GM,GCF,GCS,RM,BC,OM,OY)
bray_dist = as.matrix(vegdist(t(norm_seqtab)[sample_name[samples],], method = "bray"))

# save the plot to a folder
setwd("C:/users/ssalp/OneDrive/Documents/Advanced Microbiology and Metagenomics/Images for Report")
png(paste(format("heatmap3"), "png", sep="."), width=12, height=10,
units="in", res=300) #heatmap1.png"

# Visualise the distance matrix as a clustered heatmap
pheatmap(
  bray_dist,
  clustering_distance_rows = as.dist(bray_dist),
  clustering_distance_cols = as.dist(bray_dist),
  labels_row = paste(samples, sample_type[samples]),
  labels_col = paste(samples, sample_type[samples]),
)
dev.off()
```