Marius Zeeb, Jessy Duran Ramirez
13th December 2022 Marius.Zeeb@usz.ch

Reconstruction of microbiome using 16S ribosomal RNA NGS Data
Exercise sheet

GOALS: We learn to assess the microbiome from patient samples by analyzing data from a public data set of patients with HIV with or without tuberculosis. The data is from the publication provided to you (”hiv_tb_lung_microbiom.pdf”). 16S rRNA is a highly conserved region in bacterial genomes. It is therefore very ”easy” to isolate. Moreover, within the 16S rRNA are highly variable regions which can be used to distinguish between bacterial taxa.

Download the required files from the following link: Required is only the basic_download, it has eight samples included. The extended_download contains all the 35 samples if you are interested. However, depending on how many and which sample, the runtime will be greatly increased. ”https://filesender.switch.ch/filesender2/?s= downloadtoken=8a427a91-03f1-4cdb-9a2a-e9a8d87a3ae8”

Problem 1: NGS data

Objectives

In this first ”problem” we learn about NGS data.

1. Set the datapath were you stored the downloaded fastq files (put them in a seperate folder if not already).

```
path_to_files = "Your data path for fastq files"
list.files(path_to_files)
```

2. Read in the first file:

   We need the bioconductor package ”shortRead”. Please install and load it.

```
install.packages(tidyverse)
BiocManager::install("ShortRead")
library(tidyverse)
library(ShortRead)
fastq = readFastq(paste0(path_to_files,list.files(path_to_files)[1]))
```

3. Now we can see how the actual stored data looks:

```
fastq@sread
```

   Here you see a subset of all the stored ”reads”, i.e., sequences of nucleotides, which are the result of NGS sequencing.

   In contrast to a fasta file, were only sequences are stored, fastq files store the quality of every position of every read as well (Q score)

```
fastq@quality@quality
```

The scores are encoded in ASCII characters, i.e., each character represents a number, from 33 (!) to 73 (I). This translates into a Q score from Q0 (0% accuracy of the read position) to Q40 (99.99% accuracy of the read position)

4. Forward-/reverse reads

   As you may have noticed each file you downloaded has two version. "R1" and "R2" These are paired reads were R2 is the complementary sequence of R1. Also often called forward and reverse reads.

Problem 2: Extracting microbiom from all samples

Objectives

Now that we know what a fastq file is we can classify every read in every sample to its taxa, i.e., the bacteria from which the read/sequences in the sample originates.

We will use the pipeline from the "dada2" R package.

1. **Install/load required libraries and specify the datapaths of the forward and reverse reads:**

   ```
   BiocManager::install("dada2")
   BiocManager::install("phyloseq")
   library(dada2)
   library(phyloseq)
   ```

   Forward reads:

   ```
   fnFs = sort(list.files(path_to_files, pattern="R1_001.fastq", full.names = TRUE))
   ```

   Now the same with the reverse read files:

   ```
   fnRs =
   ```

   Extract the sample names (the naming is not fully uniform: run the following lines, we will need this later

   ```
   sample_names = gsub("JG-JG-","JG-",fnFs)
   sample_names  = sapply(strsplit(sample_names, "-"), `[`, 2)
   sample_names  = sapply(strsplit(sample_names, "_"), `[`, 1)
   ```

2. **Read the metadata:**

   Please load the provided meta data and format the samples names.

   ```
   meta_dat = read.csv("your data path to metadata") %>% dplyr::filter(id != "")
   ```

3. **Specify filtered datapaths for later:**

   Specify a new datapath for the filtered files, the folder "filtered" will be added to that path

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

4. **Select a subset of all samples:**

```
I strongly advise to not use all samples (35) choose 4 of each group. You can filter them like the following.
Look at the metadata, an choose four ”Lab.ID” of each ”TB group”.

metadata_use = meta_dat %>% filter(Lab.ID %in% c(pick your ids, for of each group))

fnFs = fnFs[names(filtFs) %in% metadata_use$Lab.ID]
filtFs = filtFs[names(filtFs) %in% metadata_use$Lab.ID]
fnRs =fnRs[names(filtRs) %in% metadata_use$Lab.ID]
filtRs = filtRs[names(filtRs) %in% metadata_use$Lab.ID]
```

5. **Quality/Q-score of reads:**

   The following functions will plot the read position over the quality score (Q-score).

```
plotQualityProfile(fnFs)
plotQualityProfile(fnRs)
```

   As you can see in the beginning the reads have a very high Q-score. Since very low Q-scores can bias our analysis we want to remove those.

6. **Clean fastq files:**

   With the following function you can clean your fastq files. It takes the orignial samples, filters them, and stores them in the prespecified filter data paths above.

   Choose fitting values:

   ”trunclen” truncates the sequences at the specified position. Based on the previous plots choose a threshold were you want to truncate the reads if the quality becomes to low. Choose this value (separate for forward and reverse reads).

   ”truncQ” truncates reads at the first occurence of Q-score equal to or below the specified value. Choose this value

   ”maxN” defines how many non-defined bases (”N”) are allowed, leave it at 0.

   ”maxEE” removes reads with more than the specified expected errors, leave it at two.

   ”rm.phix” phi-genome is a common control for 16S runs, this parameter removes reads which belong to it. Leave it at True.

   There are many additional parameters of the function feel free to play around.

   When you look at the resulting data you see how many reads were filtered out. Change parameters if you loose everything.

```
out = filterAndTrim(fnFs, filtFs, fnRs, filtRs, truncLen= c(??,??),
                     maxN=0, maxEE=2, truncQ=??, rm.phix=TRUE,
                     compress=TRUE, multithread=TRUE) # On Windows set multithread=FALSE
head(out)
```

   There are many additional parameters of the function feel free to play around.

   When you look at the resulting data you see how many reads were filtered out.

7. **Error calculation:**

   We will now estimate the errors of base calling: The function estimates the correct base at each by infering the different OTUs (operational taxonomic unit) within the samples and flags wrong bases based on the consensus sequence.

   Can you understand the ouput of the following figures?

```
errR = learnErrors(filtRs, multithread=TRUE)
errF = learnErrors(filtFs, multithread=TRUE)
plotErrors(errR, nominalQ=TRUE)
plotErrors(errF, nominalQ=TRUE)
```

8. **Extract sample composition:**

   Now we can finally extract the different taxa which are in our sequence:

   ```
   dadaRs = dada(filtRs, err = errR, multithread=TRUE)
   dadaFs = dada(filtFs, err= errF, multithread=TRUE)
   ```

9. **Final cleaning and data formatting:**

   The next function combines the forwards and reverse samples.

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

   Here we generate a table, with rows as samples, columns as unique sequence (not classifed to taxa yet) and the value is the read count of the specific sequence in the respective sample.

   ```
   seqtab = makeSequenceTable(mergers)
   ```

   It is very likely that a lot of ”thrash” is sequenced as well. The following function recognizes chimeric sequences and removes those. This will probably be a major part of the sequences. This step may take a while...get a coffee...or two.

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

Problem 3: Analysis of extracted microbiome

Objectives

Now that we have sequences and the respective counts for each samples, we want to know what bacteria is behind each sequence.

1. **Classify sequence:** Here the taxa is only estimated via bayesian/conditional probabilities.

   You the need provided file which stores the taxa and sequences ”silva_nr99_v138.1_train_set.fa”.

   The ”tryRC” parameter checks both read directions in case our sequences are in the reverse compared to the database file.

   ```
   taxa = assignTaxonomy(seqtab.nochim,"path to silva_nr99_v138.1_train_set.fa",
   multithread=TRUE,tryRC=TRUE)
   ```

   Because the assignment by probabilities is error prone. The following function will check each read position with the database (silva_species_assignment_v138.1.fa) for an exact assignment

   ```
   taxa = addSpecies(taxa, "path to silva_species_assignment_v138.1.fa")
   ```

   The resulting data shows you each sequence and its taxonomic classification

```
View(taxa)
```

Problem 4: Analysis of microbiome abundance by TB status

Objectives

Now that we have actual readable data, lets compare the two groups.

1. **Bunch of formatting first:**

   Behind each sample is a patient with or without TB:

   ```
   ##Get sample names as subject id
   subject  =  rownames(seqtab.nochim)
   ##Filter meta data to relevant samples
   meta_dat = meta_dat %>% filter(Lab.ID %in% subject)
   ##Order metadata to match order in Sequence data.
   meta_dat = meta_dat[match(subject, meta_dat$Lab.ID),]

   ##Add TB status
   tb = meta_dat$TB

   ##Generate new data frame with patient id and TB status
   meta = data.frame(Subject=subject, tbstatus=tb)

   ##Add ids as rownames
   rownames(meta) =  rownames(seqtab.nochim)

   ##TB status as factor
   meta$tbstatus = as.factor(meta$tbstatus)

   ##Build combined data with meta data, sequence data, and taxonomic data
   finaldata = phyloseq(otu_table(seqtab.nochim, taxa_are_rows=FALSE),
                  sample_data(meta),
                  tax_table(taxa))
   ```

2. **Abundance of top xxx taxa in each sample:**

   Here we plot the top most common taxa, depending on the sample choose a value between 20 and 400. You may need to experiment.

   ```
   topxx = names(sort(taxa_sums(finaldata), decreasing=TRUE))[1:...]
   topxx = topxx[!is.na(topxx)]
   finaldata.topxx = transform_sample_counts(finaldata, function(OTU) OTU/sum(OTU))
   finaldata.topxx = prune_taxa(topxx, finaldata.topxx)
   plot_bar(finaldata.topxx, fill="Order") + facet_wrap(~tbstatus, scales="free_x")
   ```

3. **Diversity index:**

   We will look at two measurements of diversity. The Shannon diversity: which shows the uncertainty if a taxa prediction of a random samples would be correct.

   The Simpson diversity: The probability that two random samples belong to the same taxa. (Figure shows 1 - Simpson diversity))

```
plot_richness(finaldata, x="tbstatus", measures=c("Shannon","Simpson"))
```

Question 1: calculate the statistical difference between the two groups, use the functions below

```
estimate_richness()
wilcox.test()
```

4. **Abundance of specific taxa in each sample:**
   Question 2: look for abundance of taxa of interest
   Specify a taxa of interest on a specific taxa level. Multiple taxas of interest can be assigned, split them with the "or" character.

```
taxa_of_interest = c("...|...|...")
taxa_level = "..."
spec_taxa = rownames(finaldata@tax_table[grep(taxa_of_interest,
finaldata@tax_table[,taxa_level]),])
spec_taxa = spec_taxa[!is.na(spec_taxa)]
finaldata.spec_taxa = transform_sample_counts(finaldata, function(OTU) OTU/sum(OTU))
finaldata.spec_taxa = prune_taxa(spec_taxa, finaldata.spec_taxa)
plot_bar(finaldata.spec_taxa, fill="Order") + facet_wrap(~tbstatus, scales="free_x")
```

   Question 3: Can you find out which group has tuberculosis?
   You may not be able to find tuberculosis in each sample of the TB group or maybe not at all. If thats the case, can you still guess the group based on the topxxx figure or the diversity figure?

Problem 5: Data visualisation via phylogeny

Objectives

Use the given code to build a phylogenetic tree.

Additional libraries required

```
BiocManager::install("DECIPHER")
BiocManager::install("ggtree")
BiocManager::install("ggtreeExtra")
install.packages(tidyverse)
install.packages("phangorn")
install.packages("ggnewscale")
library(DECIPER)
library(ggtree)
library(ggtreeExtra)
library(tidyverse)
library(phangorn)
library(ggnewscale)
```

1. **Reformat data:**
   Calculate data wide abundance for each OTU (operational taxonomic unit)

```
finaldata = transform_sample_counts(finaldata, function(OTU) OTU/sum(OTU))
```

Reformat data into a data frame (long format) and recode sample name.

```
finaldata_melt = psmelt(finaldata)
finaldata_melt$Sample = paste0("pat_",as.factor(finaldata_melt$Sample))
```

Filter the samples according to Taxa, Abundance, Samples, TB status etc. (No right or wrong) Some examples are given.

```
finaldata_melt = finaldata_melt %>% filter(!is.na(Phylum)) %>% filter(Abundance > 0.001)
  #group_by(Phylum,Class,Order,Family,Genus) %>%
  #mutate(n = sum(Abundance > 0)) %>% #filter(n > 1)
```

2. **Build phylogenetic tree:**

   Build a distance matrix from all the sequences

```
##Extract sequences
sequences=getSequences(unique(finaldata_melt$OTU))
##Sequence name same as sequence to match later
names(sequences)=sequences
#Alignment of sequences
alignment = AlignSeqs(DNAStringSet(sequences), anchor=NA)
phang.align = phyDat(as(alignment, "matrix"), type="DNA")
```

Tree building

```
##dist.ml tries different variants of genetic distance measures and
##takes the best fitting to the actual data
dm = dist.ml(phang.align)
##Neighbour joining is used for the tree construction
treeNJ = NJ(dm)
```

Usually one uses resampling methods to construct many trees and then taking the one with the highest likelihood. For the sake of time you can skip this (depending on your sequence-sample size this might take a long time). But if you like

```
fit = pml(treeNJ, data=phang.align)
fitbsub = update(fit, k=4, inv=0.2)
bestsubmodel = ## assign as string what written under model when you type "fit" in the console
fitbsub = optim.pml(fitbsub, model=bestsubmodel, optInv=TRUE, optGamma=TRUE,
          rearrangement = "stochastic", control = pml.control(trace = 0))
```

3. **Visualize phylogenetic tree:**

   Now you are ready to plot the tree. (Depending on you filtering steps, changes below might be required, e.g., if you only want to plot non-tb sample, remove the second geom_fruit)

```
p = ggtree(treeNJ, layout="fan", open.angle=15)

p = p %<+% finaldata_melt

p = p + new_scale_fill() + new_scale_color() +
  geom_fruit(geom=geom_tile,
             mapping=aes(y=OTU, x=Phylum, fill=Phylum),
             offset = 0.04,size = 10) +
  scale_fill_hue() +
  new_scale_fill() + new_scale_color() +
  geom_fruit(geom=geom_tile,
             mapping=aes(y=OTU, x=tbstatus, fill=Abundance),
              offset = 0.04,size = 0.02) +
  scale_fill_viridis_c() +
  new_scale_fill() + new_scale_color() +
  geom_fruit(geom=geom_tile,
             mapping=aes(y=OTU, x=Sample, fill=Sample),
              offset = 0.1,size = 0.02) +
  scale_fill_manual(values=c("#FFC125","#7B68EE",
                               "#800080","#D15FEE",
                               "#EE6A50","#006400",
                               "#800000","#B0171F"))

p = p +  geom_treescale(fontsize=2, linesize=1, x=1, y=1) +
          theme(legend.position=c(1.1, 0.5),
          legend.background=element_rect(fill=NA),
          legend.spacing.y = unit(0.1, "cm"),
  )
p
```