Processing 16S rRNA gene amplicons with DADA2

Brazelton Lab
October 2019
Primary author was Christopher Thornton, with contributions from Shahrzad Motamedi, Cody Dangerfield, and Julia McGonigle

Note: The DADA2 tutorial and manual are very helpful references and sources for the workflow below.

Contents

References

Callahan JB, McMurdie PJ, Rosen MJ, Han AW, Johnson AJ, Holmes SP. 2016. DADA2: High resolution sample inference from amplicon data. *Nature Methods*. doi: 10.1038/nmeth.3869.

Cole JR, Wang Q, Fish JA, Chai B, McGarrell DA, Sun Y, Brown CT, Porras-Alfaro A, Kuske CR, Tiedje JM. 2014. Ribosomal Database Project: data and tools for high throughput rRNA analysis. *Nucl. Acids Res.* 42(D1):D633-D642. doi: 10.1093/nar/gkt1244.

DeSantis TZ, Hugenholtz P, Larsen N, Rojas M, Brodie EL, Keller K, Huber T, Dalevi D, Hu P, Andersen GL. 2006. Greengenes, a Chimera-Checked 16S rRNA Gene Database and Workbench Compatible with ARB. *Appl. Environ. Microbiol.* 72:5069-72.

Martin M. 2011. Cutadapt removes adapter sequences from high-throughput sequencing reads. *EMBnet.journal* 17(1): 10-12. doi: 10.14806/ej.17.1.200.

McMurdie PJ, Holmes S. 2013. phyloseq: an R package for reproducible interactive analysis and graphics of microbiome census data. *PloS One* 8(4), e61217. doi:10.1371/journal.pone.0061217.

Wang Q, Garrity GM, Tiedje JM, Cole JR. 2007. Naive Bayesian classifier for rapid assignment of rRNA sequences into the new bacterial taxonomy. *Appl. Environ. Microbiol.* 73: 5261–5267. doi: 10.1128/AEM.00062-07.

Data Preparation

Starting Materials: Raw Illumina paired-end reads with partial to complete overlap in FASTQ format

Make a new project directory and create links to the appropriate raw data files:

```
mkdir -p <project>/short_reads
```

```
cd <project>/short_reads
ln -svi /path/to/fastq_files/* ./
```

Generate basic read statistics for each sample and remove any failed libraries:

```
srun readstats.py --csv -o readstats.csv ./* &
less readstats.csv
mkdir failed_libs
mv ./<files_with_very_low_counts> failed_libs/
```

Create a workspace to process the data in:

```
mkdir ../preprocess
cd ../preprocess
```

Discard contaminant sequences:

a. Create a slurm batch file named decontam.sh with the following content (replace **<text>** with the appropriate information):

```
#! /bin/sh
#SBATCH -J <project>.decontam
reads_path="</path/to/short_reads>";
files=$(find -L ${reads_path} -type f -regex ".*\.20.forward\.fastq.*");
for file in ${files[@]}; do
   file=$(basename ${file});
   IFS=". " read -a array <<< ${file};
   sample=${array[0]};
   forward="${reads_path}/${sample}.20.forward.fastq.gz";
   reverse="${reads_path}/${sample}.20.reverse.fastq.gz";
   cutadapt --discard-trimmed --error-rate 0.10 -a ATTAGAWACCCVHGTAGTCCGGCTGACTGACT -A
TTACCGCGGCMGCTGGCACACAATTACCATA -g ^TTAGAWACCCVHGTAGTCCGGCTGACTGACT -G
^TACCGCGGCMGCTGGCACACAATTACCATA -o ${sample}.20.forward.decontam.fastq.gz -p $
{sample}.20.reverse.decontam.fastq.gz ${forward} ${reverse} > ${sample}.cutadapt.log
done
```

b. Execute the batch script

```
sbatch decontam.sh
```

* in amplicon sequence data, contaminants tend to come from the primer sequences used to amplify a region of the genome.
* assumes primers 515f and 806r as described in Kozich et al, 2013 were used to amplify the V4 region of the 16S rRNA gene. If not, the appropriate sequences will need to be provided to cutadapt arguments -a/-A and -g/-G. Use fastqc to determine potential adapter and primer sequences that need to be removed. See srun cutadapt --help for details on usage. Note that forward primers tend to be found in reverse reads and reverse primers in forward reads.

Confirm that most of the reads in each sample passed the sequence contamination filters:

```
grep "Pairs written (passing filters):" *.log > passed_filter.txt
less passed_filter.txt
```

* if only a low percentage passed, you may consider removing the sample from the dataset.

Create a screen session for the project:

```
screen -S <project>
```

Start R and load the required libraries:

```
srun --x11=first --pty R
library('dada2')
library('ShortRead')
library('ggplot2')
library('grid')
library('gridExtra')
```

Set the appropriate path variables:

```
path <- getwd()
reads <- list.files(path)
```

Select only the fastq files and sort them so that the forward and reverse reads are in the same order:

```
fastqs <- sort(reads[grepl('.fastq+', reads)])
ff <- fastqs[grepl('forward', fastqs)]
rf <- fastqs[grepl('reverse', fastqs)]
```

Obtain a list of sample names:

```
samples <- sapply(strsplit(ff, '[.]'), '[', 1)
```

* takes the n^{th} item of a string after being split by a delimiter. Assumes that files are delimited by a period, with the sample name as the prefix (e.g. 0AMRd001.forward.fastq). If a character other than a period is used as the delimiter, insert the appropriate character into the second argument of strsplit.

Add full paths to the files:

```
ff <- paste0(path, '/', ff)
rf <- paste0(path, '/', rf)
```

Examine a sampling of the dataset's quality profile:

```
n <- sample(length(ff), <num_samples>)
forward_plots <- list()
reverse_plots <- list()
for (i in 1:length(n)) {
  sample_index <- n[i]
  fp <- plotQualityProfile(ff[sample_index])
  rp <- plotQualityProfile(rf[sample_index])
  fp <- fp + ggtitle(samples[sample_index])
  rp <- rp + ggtitle(samples[sample_index])
  forward_plots[[i]] <- fp
  reverse_plots[[i]] <- rp
}
grid.arrange(grobs=forward_plots)
grid.arrange(grobs=reverse_plots)
```

* the diagrams show the distribution of quality scores as a function of sequence position. Take note of where the drop in quality tends to occur, as this information will be used in later preprocessing steps. If it is not clear from using num_samples samples, additional samples can be plotted to get a better idea of the quality trends of the dataset as a whole.
* the plotted lines are the summary statistics for each position: solid green reflects the mean, solid orange the median, and dashed orange the 25th and 75th quantiles.

Set additional path variables:

```
ff.filt <- paste0(path, '/', samples, '.forward.decontam.filtered.fastq.gz')
rf.filt <- paste0(path, '/', samples, '.reverse.decontam.filtered.fastq.gz')
```

Preprocessing

Note: Preprocessing should be done separately when samples are from different sequencing runs and then merged later, if desired.

Trim sequences and filter by quality:

```
filtered <- filterAndTrim(ff, ff.filt, rf, rf.filt, maxN=0, maxEE=3, truncQ=2,
compress=TRUE, verbose=TRUE, matchIDs=TRUE, rm.phix=c(TRUE, TRUE))
```

Combine identical sequences to retain only the uniques:

```
ff.derep <- derepFastq(ff.filt, verbose=TRUE)
rf.derep <- derepFastq(rf.filt, verbose=TRUE)
```

Add sample names to the derep-class objects:

```
names(ff.derep) <- samples
names(rf.derep) <- samples
```

Estimate error rates to be used in the dada algorithm:

```
ff.err <- learnErrors(ff.derep[n], errorEstimationFunction=loessErrfun,
randomize=FALSE)
rf.err <- learnErrors(rf.derep[n], errorEstimationFunction=loessErrfun,
randomize=FALSE)
```

* n, the same sample subset that was generated above, should be large enough to adequately represent the dataset as a whole. It is recommended to start with a small fraction of the total sample size (~10% for large datasets, 20-30% for small) and then use the error plots generated in the next step to determine if the rates have been sufficiently estimated. If not, generate a new subset with n <- sample(length(ff), <**num_samples**>) and rerun the command.

Visualize the estimated error rates:

```
plotErrors(ff.err, nominalQ=TRUE)
plotErrors(rf.err, nominalQ=TRUE)
```

* plots the observed frequency of each transition and transversion (observed error rates) as a function of quality score. The red line represents the expected error rates. The black line represents the fitted rates after convergence.

* the plots should be used to verify that the error rates have been sufficiently estimated. Visually, this can be determined by how well the model (black line) tracks the observed error rates (the black dots). Note that this does not mean that the line needs to pass through all the dots; often a more general model is a better predictor than one that fits the data extremely well. If the error model does not provide a sufficient fit, re-run the parameter estimation step with a larger sample subset.

Infer sequence variants from the dataset:

```
ff.dada <- dada(ff.derep, err=ff.err, pool=TRUE)
rf.dada <- dada(rf.derep, err=rf.err, pool=TRUE)
```

* samples should be pooled (pool=TRUE) whenever possible as it allows information to be shared across samples, making it easier to resolve rare variants. Samples should not be pooled, however, unless they share a similar error 'history'. Cases in which samples will not have a shared error history include when they come from different sequencing runs or if different PCR protocols were used for amplification. In these cases, dada should be run with the parameter selfConsist=TRUE or separately for the different sample sets. The memory requirements are also larger for pooled samples, so it may be necessary to run the command with selfConsist=TRUE for very large datasets (>200 samples).

Inspect the dada-class objects:

```
ff.dada
rf.dada
```

* see help('dada-class') for the kinds of information accessible from dada-class objects.

Merge overlapping paired-end reads:

merged <- mergePairs(ff.dada, ff.derep, rf.dada, rf.derep, verbose=TRUE, maxMismatch=0, trimOverhang=FALSE, justConcatenate=FALSE)
head(merged[[1]])
head(merged[[2]])

Construct a sequence-by-sample table:

```
seqtable <- makeSequenceTable(merged, orderBy='abundance')
dim(seqtable)
table(nchar(colnames(seqtable)))
```

Remove chimeric sequences:

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

Determine how many total sequences are retained after each step:

```
getN <- function(x) sum(getUniques(x))
track <- cbind(filtered, sapply(ff.dada, getN), sapply(merged, getN),
rowSums(seqtable.nochim))
colnames(track) <- c("raw", "q-filtered", "denoised", "merged", "chimera-checked")
rownames(track) <- samples
track
```

Analysis

Create a workspace to analyse the data:

```
path <- '/path/to/project_dir/analysis/'
system(paste('mkdir', path))
```

Assign taxonomy to the inferred, chimera-filtered sequences:

```
refdb <- /srv/databases/markers/silva/dada2/silva_nr_v132_train_set.fa.gz
taxa <- assignTaxonomy(seqtable.nochim, refdb, tryRC=FALSE, minBoot=50, verbose=TRUE)
colnames(taxa) <- c("Domain", "Phylum", "Class", "Order", "Family", "Genus")
```

Merge the different components of the data into a phyloseq-class object:

```
library('phyloseq')
meta.table <- read.table("<meta-data table>", sep=',', header=TRUE, row.names=1)
<project>.dat <- phyloseq(otu_table(seqtable.nochim, taxa_are_rows=FALSE),
tax_table(taxa), sample_data(meta.table))
```

Output processed data to files:

```
fasta.out <- paste0(path, '/<project>.preprocessed.fasta')
seqtable.out <- paste0(path, '/<project>.seqtable.tsv')
taxa.out <- paste0(path, '/<project>.taxonomy.tsv')
unqs <- getUniques(seqtable.nochim)
uniquesToFasta(unqs, fasta.out)
ids <- paste0("sq", seq(1, length(unqs)), ";size=", unname(unqs), ";")
seqtable.tmp <- t(seqtable.nochim)
row.names(seqtable.tmp) <- ids
write.table(seqtable.tmp, file=seqtable.out, quote=FALSE, sep="\t", row.names=TRUE,
col.names=TRUE)
taxa.tmp <- taxa
row.names(taxa.tmp) <- ids
write.table(taxa.tmp, file=taxa.out, quote=FALSE, sep="\t", row.names=TRUE,
col.names=TRUE)
```

Merge the taxonomy and otu table into one file for convenient browsing:

```
srun count_cat_tax_csv.py -t tax-table.csv -c otu-table.csv -o otu-tax-table.csv
```