16S Amplicons Processing

Instructions for using DADA2 to process high-throughput sequences of 16S rRNA gene hypervariable region amplicons

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

Note: Familiarize yourself with the workflow on the training dataset provided in /srv/data/training/amplicons/mothur/mothur_miseq_sop prior to following it for the first time on real data.

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.

Quast C, Pruesse E, Yilmaz P, Gerken J, Schweer T, Yarza P, Peplies J, Glöckner FO. 2013. The SILVA ribosomal RNA gene database project: improved data processing and web-based tools. *Nucl. Acids Res. 41 (D1): D590-D596.*

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/* ./
```

Important note: If your filenames have important information after the first dot (e.g. .20 and .21 or .20A and .20B, then you will want to change these dots to underscores when you create the symbolic links.)

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.
* the linux utility `less` can be used to quickly search if a file contains a given sequence (i.e. a primer sequence).

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')
```

* X11 forwarding should be enabled when connecting to the cluster in order to view the diagrams generated in this workflow. That, or diagrams can be saved to a file and opened on a local machine.

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 nth 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. 0RBCd001.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, truncLen=c(<Flength>, <Rlength>),
trimLeft=c(<Fbases>, <Rbases>), maxN=0, maxEE=c(<Ferr>, <Rerr>), truncQ=2,
compress=TRUE, verbose=TRUE, matchIDs=TRUE, rm.phix=c(TRUE, TRUE))
```
* sequence quality tends to drop drastically towards the 3'-end of a read when sequencing is performed on Illumina platforms. It is generally a good idea to truncate sequences to where the crash in quality occurs prior to quality-based filtering. The number of bases to trim will depend not only on the quality of the reads, but also template and read lengths. Removing too many bases from either strand may prevent paired reads from merging. However, more errors may be found in a sequence if too few bases are trimmed. It is important to know the size of the DNA fragments sequenced as well as the size of the reads generated by the sequencing platform. Enough bases should be retained to allow for a decent amount of overlap between the strands (i.e. 40-60 bases).
* the authors of DADA suggest that the first 10 bases from Illumina-generated sequences be removed, as the error rates are less controlled at the start of sequencing. Keep in mind that while removing bases from the ends of paired reads should not affect the overall size of the merged sequence, removing bases from the 5'-end will reduce its length and thus the number of characters that can be used to distinguish variants; I recommend only trimming bases from the start of a read if there is a noticeable difference in quality from the rest of the quality distribution.
* the `maxEE` parameter specifies the maximum number of expected errors allowed in a single read. This value should be kept low (recommended value: 2-3 bases).
* the following example trims reads to a specified length, without trimming bases from the 5' end: `filtered <- filterAndTrim(ff, ff.filt, rf, rf.filt, truncLen=c(220, 160), maxN=0, maxEE=c(2, 2), truncQ=2, compress=TRUE, verbose=TRUE, matchIDs=TRUE, rm.phix=c(TRUE, TRUE))`
* **new** DADA2 now supports input reads of different length. Which means that trimming all reads to an equal length is no longer required. This will allow more flexibility in how reads can be preprocessed, so feel free to use a different method for quality control of raw reads if preferred.

Combine identical sequences to retain only the uniques:
```
ff.derep <- derepFastq(ff.filt, verbose=TRUE)
rf.derep <- derepFastq(rf.filt, verbose=TRUE)
```
* a consensus quality score and abundance information is stored along with the unique sequences.
* see `help('derep-class')` for the kinds of information accessible from derep-class objects.

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)
```

* the denoising algorithm is built on a model of errors in Illumina reads. It quantifies the rate at which an amplicon read is produced from a sample sequence as a function of both sequence frequency and quality.
* 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,
minOverlap=<length>, maxMismatch=<bases>, trimOverhang=FALSE, justConcatenate=FALSE)
head(merged[[1]])
head(merged[[2]])
```

* (**optional**) the minimum overlap length (`minOverlap`) will depend on both the size of the region sequenced (fragment size) and the size of the reads generated by the sequencing technology. An approximation of the overlap length can be obtained by overlap_len ~= (2 * read_len) - frag_len. If trimming was performed, the number of bases removed from both the forward and reverse strands will also need to be factored into the calculation. Some deviation from the expected length should be allowed, as variation in sequence size may have a biological basis.
* `maxMismatch` is the maximum number of differences between base pairs allowed in the overlapping region. For example, to allow a 5% difference: maxMismatch = estimated_overlap_length * 0.05. The authors of DADA2 recommend that paired-end reads only be merged if they have an exactly overlapping region. Since merging is performed after denoising it is expected that nearly all substitution errors have already been removed at this point. To disallow discrepant positions, either leave off the `maxMismatch` parameter or set `maxMismatch` to `0`. I recommend basing the decision on whether to allow mismatches on the amount of overlap remaining after preprocessing. If the amount of overlap is low, then disallowing mismatches is probably best. But if the amount of overlap is high, as would be the case if no bases were cropped from either strand (~247 overlapping bases), then I would allow one or two discrepant positions.

Construct a sequence-by-sample table:
```
seqtable <- makeSequenceTable(merged, orderBy='abundance')
dim(seqtable)
table(nchar(colnames(seqtable)))
```
* a sequence-by-sample table is similar to an OTU table produced from OTU methods.
* the `dim` function outputs the dimensions of the sequence table, where rows (first item) are samples and columns (second item) are sequences.
* the table function outputs a table of sequence length by number of sequences with that length. Most sequences of the V4 hypervariable region should be around 253 bp in length unless bases were removed from the 5'-end with the --headcrop option during trimming.

Remove chimeric sequences:
```
seqtable.nochim <- removeBimeraDenovo(seqtable, method="consensus", verbose=TRUE)
dim(seqtable.nochim)
```
* identifies all sequences that can be reconstructed as a two-parent chimera from more abundant sequences.
* the fraction of total reads found to be chimeric can be obtained from `1 - sum(seqtable.nochim)/sum(seqtable)`. The fraction of all unique inferred sequence variants found to be chimeric can be obtained with 1 - `ncol(seqtable.nochim)/ncol(seqtable)`

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 <- '/path/to/reference_database'
taxa <- assignTaxonomy(seqtable.nochim, refdb, tryRC=FALSE, minBoot=50, verbose=TRUE)
colnames(taxa) <- c("Domain", "Phylum", "Class", "Order", "Family", "Genus")
```

* `assignTaxonomy` uses an implementation of the RDP naive Bayesian classifier.
* available reference databases are SILVA, GreenGenes, and RDP. The databases are located on the server at /srv/databases/markers/<db_name>/dada2/.

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
```

Continue analysis with '16S data exploration', starting after the first step

Data Transformations

Note: Certain analysis methods require untransformed count data as input. Make sure to save the transformed data as a new R object so that the original count data is not lost.
Note: Subsampling to an even depth (i.e. rarefying) isn't included here, but can be performed with the rarefy_even_depth function in phyloseq.

Define the normalization/scaling functions:
```
transform.cs <- function(seqdat, func=NULL) {
    if (is.null(func)) {
        func=median
    }
    scale <- func(sample_sums(seqdat))
    return(transform_sample_counts(seqdat, function(x, m=scale) round(m*(x/sum(x)))))
}
```
* apply a common-scale transformation. `func` should be an R function (median, mean, max, min, etc.) to use in calculating the scaling constant.

```
transform.vs <- function(seqdat, design=NULL) {
    require(DESeq2)
    # Check data table orientation
    if (taxa_are_rows(seqdat)) {
        counts <-otu_table(seqdat)
    } else {
        counts <-t(otu_table(seqdat))
    }
    #
    if (is.null(design)) {
        blind <- TRUE
        design <- "~ 1"
    } else {
        blind <- FALSE
        design <- paste("~", design)
    }
    # Add a one to each cell to avoid NaN problems
    counts <- counts + 1
    counts <- as(counts, "matrix")
    metadata <- sample_data(seqdat)
    metadata <- as(metadata, "matrix")
    dds <- DESeqDataSetFromMatrix(countData=counts, colData=metadata,
design=as.formula(design))
    # Remove sequence variants with only a single count across all samples
    dds <- dds[rowSums(counts(dds)) > 1, ]
    dds <- vst(dds, blind=blind)
    return(otu_table(assay(dds), taxa_are_rows=TRUE))
}
```
* apply a variance-stabilizing transformation (VST), as in McMurdie & Holmes, 2014. The purpose of a VST is to account for overdispersion in amplicon count data by removing dependence of the variance on the mean.
* allows samples to be grouped into experimental "conditions", which is used in estimating the amount of variability in the counts.
* the result is normalized counts on a log2 scale.

```
transform.tss <- function(seqdat) {
    return(transform_sample_counts(seqdat, function(x) x/sum(x)))
}
```
* transform counts to proportions (total-sum scaling).

```
transform.log <- function(seqdat, n=1, base=2) {
    return(transform_sample_counts(seqdat, function(x, n) log(x + n, base=base)))
}
```
* log transform count data using pseudo-counts (to avoid log(0) problems).

```
transform.css <- function(seqdat) {
    require(metagenomeSeq)
    # Check data table orientation
    if (taxa_are_rows(seqdat)) {
        counts <- as.data.frame(otu_table(seqdat))
    } else {
        counts <- as.data.frame(t(otu_table(seqdat)))
    }
    MR <- newMRexperiment(counts)
    # Calculate the percentile by which to normalize counts
    p <- cumNormStatFast(MR)
    # Normalize data by scaling counts to the pth quantile
    counts.norm <- cumNormMat(MR, p=p)
    return(otu_table(as.data.frame(counts.norm), taxa_are_rows=TRUE))
}
```
* apply a cumulative-sum scaling transformation, as in Paulson et al., 2013.

```
<project>.dat.norm <- <scaling_function>(<project>.dat)
```

Additional Filtering (optional)

Remove archaea and eukaryotes, as well as anything classified as chloroplast or mitochondria:
```
<project>.dat <- subset_taxa(<project>.dat, !Domain %in% c("Eukaryota","Archaea") &
!is.na(Domain) & !Family=="Mitochondria" & !Class=="Chloroplast"))
table(nchar(taxa_names(<project>.dat)))
```

Remove all sequences with NA at the phylum level and an unusually long or short sequence (two standard deviations from the mean):

```
seq.mean <- mean(nchar(taxa_names(<project>.dat)))
seq.sd <- sd(nchar(taxa_names(<project>.dat)))
<project>.dat <- subset_taxa(<project>.dat, !is.na(Phylum) &
nchar(taxa_names(<project>.dat)) < seq.mean + 2*seq.sd |
nchar(taxa_names(<project>.dat)) > seq.mean - 2*seq.sd)
```

Filter low prevalence, low abundance sequences:

```
filter_by_prevalence <- function(seqdat, prop) {
    # convert counts to proportions
    seqdat.ra <- transform_sample_counts(seqdat, function(x) x/sum(x))
    # find the variants found only in a single sample
    uniques <- filter_taxa(seqdat.ra, function(x) sum(x > 0) <= 1, prune=TRUE)
    # check if relative abundance of unique variants less than user defined value
    uniques.lowra <- filter_taxa(uniques, function(x) sum(x) < prop)
    bad.seqs <- names(uniques.lowra)[uniques.lowra == TRUE]
    good.seqs <- taxa_names(seqdat)[! taxa_names(seqdat) %in% bad.seqs]
    return(prune_taxa(good.seqs, seqdat))
}
ra <- 0.001
<project>.dat <- filter_by_prevalence(<project>.dat, ra)
```

* removes sequences observed only in a single sample (very low prevalence) if the proportion contributed to sample’s library size is less than the threshold defined in `ra`