DADA2 Tutorial: STAMPS 2019

Processing marker-gene data with. . .

DADA2

Important resources:

- The DADA2 website
- The DADA2 tutorial workflow
- The DADA2 Issues forum

The DADA2 Workflow

1. Preprocessing
2. Filter and Trim
3. Learn Error Rates
4. Denoise/Sample Inference
5. Merge (if paired-end)
6. Remove Chimeras
7. Assign Taxonomy

Throughout: Sanity checks!

Preprocessing, Filtering and Trimming

Preprocessing

This workflow assumes that your sequencing data meets certain criteria:

- Samples have been demultiplexed, i.e. split into individual per-sample fastq files.
- Non-biological nucleotides have been removed, e.g. primers, adapters, linkers, etc.
- If paired-end sequencing data, the forward and reverse fastq files contain reads in matched order.

See the DADA2 FAQ for tips to deal with non-demultiplexed files and primer removal.

Load package and set path

Load the dada2 package. If you don't already it, see the dada2 installation instructions:

```r
library(dada2); packageVersion("dada2")
```

```
## [1] '1.12.1'
```

```r
library(ggplot2); packageVersion("ggplot2")
```

```
## [1] '3.2.0'
```

Set the path to the fastq files:

```r
path <- "data/fastqs"
head(list.files(path))
```

```
## [1] "806rcbc288_R1.fastq.gz" "806rcbc288_R2.fastq.gz"
## [3] "806rcbc289_R1.fastq.gz" "806rcbc289_R2.fastq.gz"
## [5] "806rcbc290_R1.fastq.gz" "806rcbc290_R2.fastq.gz"
```

Forward, Reverse, Sample Names

Get matched lists of the forward and reverse fastq.gz files:

```r
# Forward and reverse fastq filenames have format: SAMPLEN
fnFs <- sort(list.files(path, pattern="_R1.fastq.gz", full
fnRs <- sort(list.files(path, pattern="_R2.fastq.gz", full
fnFs[[1]]; fnRs[[1]]
```

```
## [1] "data/fastqs/806rcbc288_R1.fastq.gz"
```

```
## [1] "data/fastqs/806rcbc288_R2.fastq.gz"
```

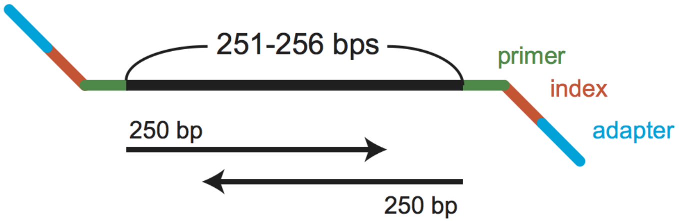
Extract sample names, assuming filenames have format: SAMPLENAME_XXX.fastq.gz

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

```
## [1] "806rcbc288" "806rcbc289" "806rcbc290" "806rcbc291"
## [6] "806rcbc293"
```

Check the amplicon design

We are using the 515F/806R primer set. The primers are not sequenced. The sequencing technology is 2x250 paired end Illumina.



What does this mean for later? Artifacts? Trimming?

What is your amplicon design?

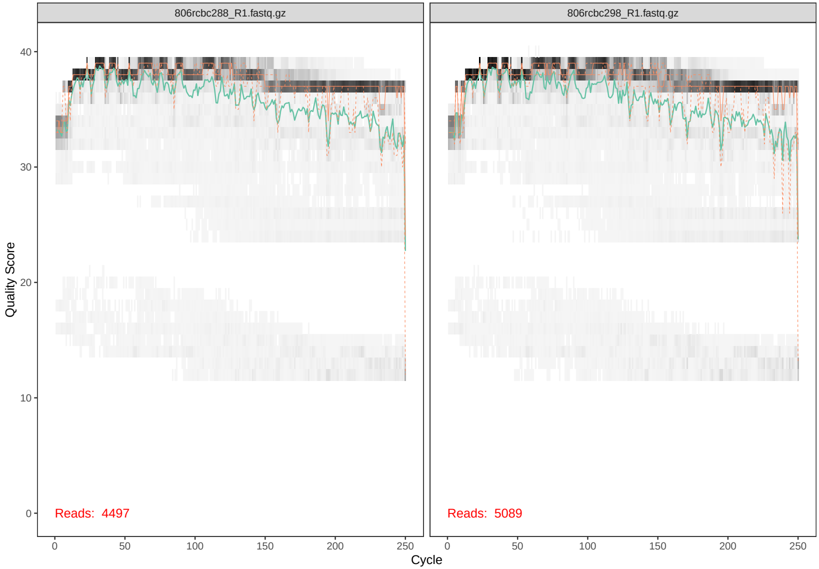
coi? trnL? amoA? ITS1? ITS2? cpn60? 18S? . . . 16S: v1v2? v1v3? v4? v3v4? v4v5? . . .

How long is it? Length variation?

Did you sequence your primers? Are you sure?

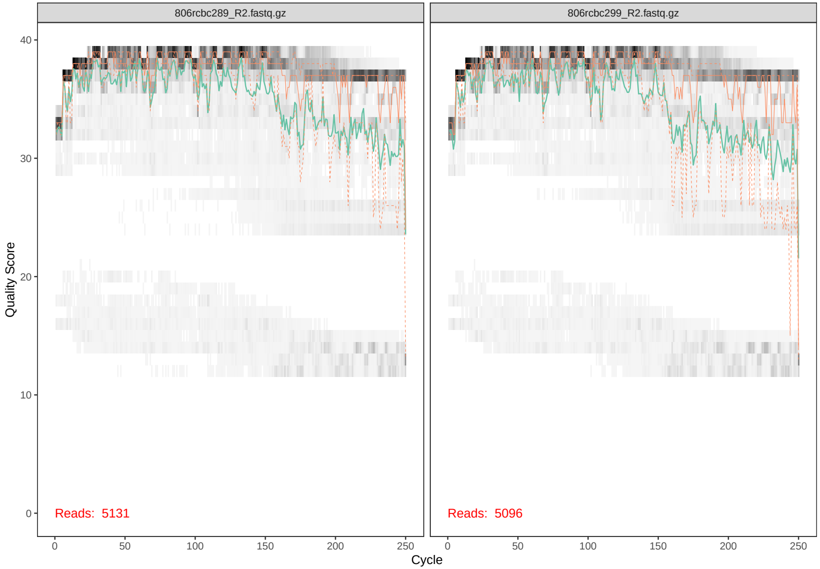
Forward quality profiles: Truncate where?

```
plotQualityProfile(fnFs[c(1,11)])
```



Reverse quality profiles: Truncate where?

```r
plotQualityProfile(fnRs[c(2,12)])
```



Filter and trim

Assign filenames for the filtered fastq.gz in the filtered/ subdirectory.

```
filtFs <- file.path(path, "filtered", paste0(sample.names,
filtRs <- file.path(path, "filtered", paste0(sample.names,
```

The critical parameters we chose are the truncation lengths of **240** (forward) and **170** (reverse). *Why did we choose these values?*

```
out <- filterAndTrim(fnFs, filtFs, fnRs, filtRs,
                     truncLen=c(240,160), maxEE=c(2,2), # r
                     compress=TRUE, multithread=FALSE) # Se
```

In most cases, the key quality filtering parameter is `maxEE`, which sets the maximum number of expected errors allowed in each read. This has been shown to be a better quality filter than an average quality score filter.

Quality filtering options

- `maxEE`: Maximum expected errors, usually the only quality filter needed.
- `truncQ`: Truncate at first occurrence of this quality score.
- `maxLen`: Remove sequences greater than this length (mostly for pyrosequencing).
- `minLen`: Remove sequences less than this length.
- `maxN`: Remove sequences with more than this many Ns. `dada2` requires no Ns, so `maxN=0` by default.
- `rm.lowcomplex`: Remove reads with complexity less than this value.

Usually `maxEE` is enough, but for non-Illumina sequencing technologies, or less standard setups, the other options can be useful as well. Remember that help is your friend! `?filterAndTrim`

SANITY CHECK: Filtering Stats

```
head(out)
```

```
##                         reads.in reads.out
## 806rcbc288_R1.fastq.gz      4497      4047
## 806rcbc289_R1.fastq.gz      5131      4638
## 806rcbc290_R1.fastq.gz      4921      4473
## 806rcbc291_R1.fastq.gz      5886      5239
## 806rcbc292_R1.fastq.gz      5116      4669
## 806rcbc293_R1.fastq.gz      5318      4755
```

- What fraction of reads were kept?
- Was that fraction reasonably connsistent among samples?
- Were enough reads kept to achieve your analysis goals?

The truncation lengths are the most likely parameters you might want to revisit.

Basic strategy: While preserving overlap of 12nts + biological length variation, truncate off quality crashes.

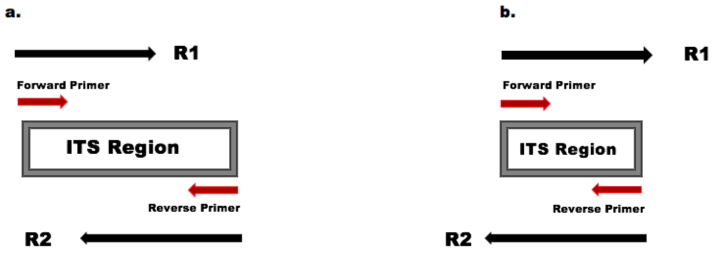
Primer removal

For common primer designs, in which a primer of fixed length is at the start of the forward (and reverse) reads, primers can be removed by dada2 in the `filterAndTrim` step.

```
# Single-end reads
filterAndTrim(..., trimLeft=FWD_PRIMER_LENGTH)
# Paired-end reads
filterAndTrim(..., trimLeft=c(FWD_PRIMER_LENGTH, REV_PRIMEI
```

However! There are other scenarios that this won't handle, in particular when amplicon length is too so variable that reads sometime read into the other primer at the end:

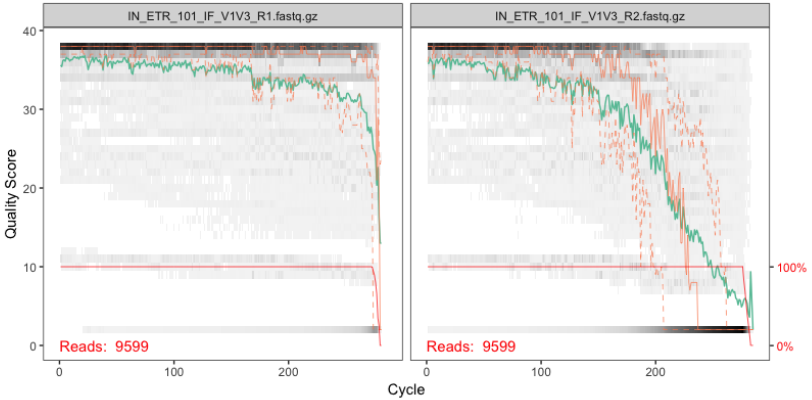
Primer removal - ITS



With highly variable amlicons, you will need to use an outside program to remove primers prior to running the dada2 workflow. If you are in that scenario, please see the DADA2 ITS workflow.

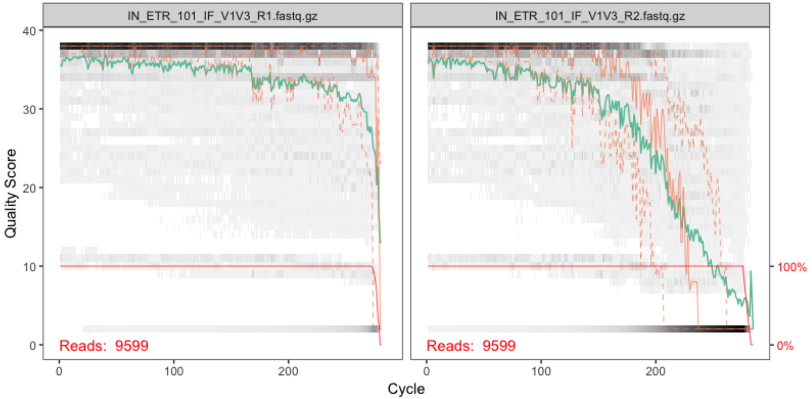
Exercise: Pick truncation and trimming values

Sequenced amplicon length: 400-420nts. Primers are sequenced.



Exercise: Pick truncation and trimming values

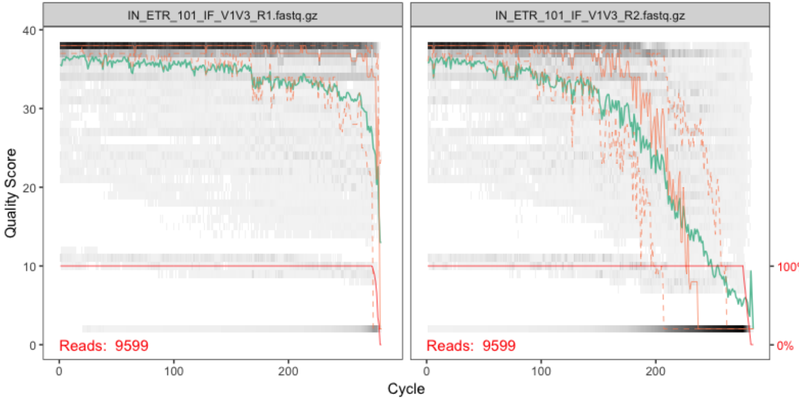
Sequenced amplicon length: 400-420nts. Primers are sequenced.



- `trimLeft=c(17, 21)`
- `truncLen=c(245, 195)`

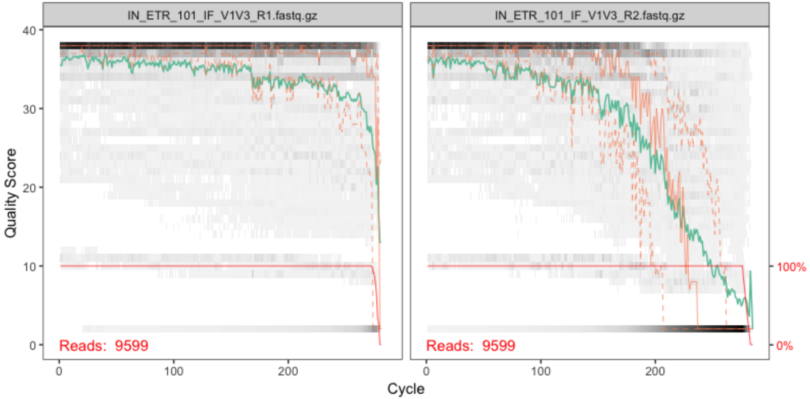
Exercise: Pick truncation and trimming values

Sequenced amplicon length: 220-320nts. Primers are not sequenced.



Exercise: Pick truncation and trimming values

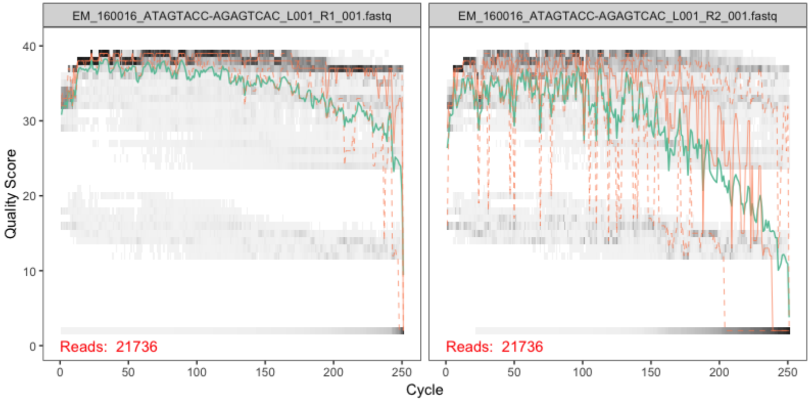
Sequenced amplicon length: 220-320nts. Primers are not sequenced.



- `trimLeft=0`
- `truncLen=c(210, 160)`

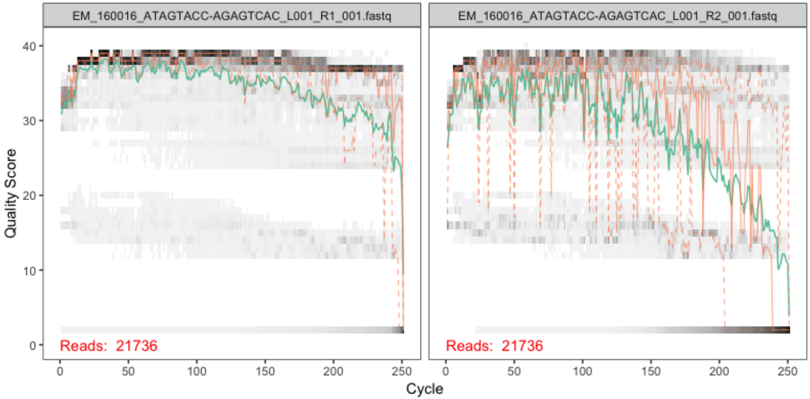
Exercise: Pick truncation and trimming values

Sequenced amplicon length: 250-260nts. Primers are sequenced.



Exercise: Pick truncation and trimming values

Sequenced amplicon length: 250-260nts. Primers are sequenced.



- `trimLeft=c(14, 17)`
- `truncLen=c(220, 140)`

Learn error rates and Denoise

Learn the Error Rates

```
errF <- learnErrors(filtFs, multithread=2) # Set multithre
```

```
## 17324880 total bases in 72187 reads from 16 samples wil
```

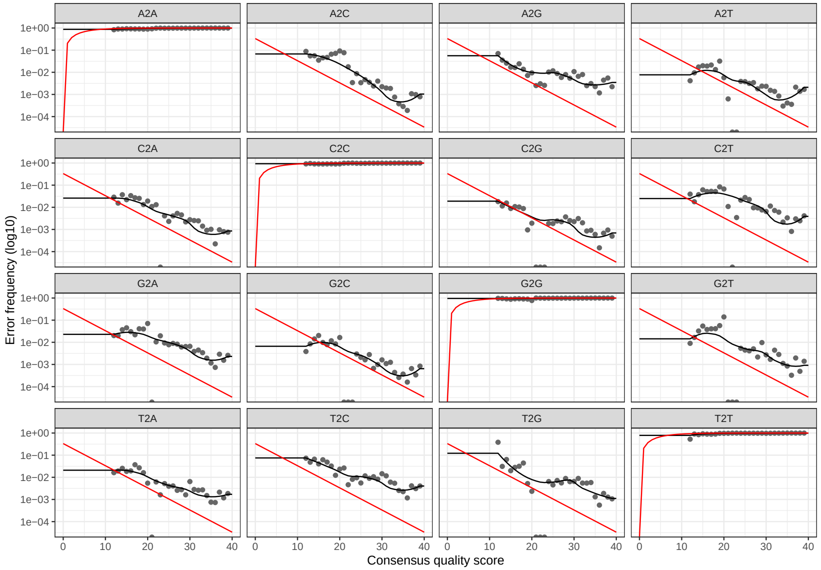
```
errR <- learnErrors(filtRs, multithread=2)
```

```
## 11549920 total bases in 72187 reads from 16 samples wil
```

The DADA2 algorithm makes use of a parametric error model (`err`) and every amplicon dataset has a different set of error rates. The `learnErrors` method learns this error model from the data, by alternating estimation of the error rates and inference of sample composition until they converge on a jointly consistent solution.

SANITY CHECK: Error Rates

```
plotErrors(errF, nominalQ=TRUE)
```



SANITY CHECK: Error Rates

- Does the model (black line) reasonably fit the observations (black points)?
- Do the error rates mostly decrease with quality score?

The goal here is good, not perfect, so don't sweat the small stuff (or non-convergence).

Dereplicate

Dereplication combines all identical sequencing reads into “unique sequences” with a corresponding “abundance” equal to the number of reads with that unique sequence.

```r
derepFs <- derepFastq(filtFs)
derepRs <- derepFastq(filtRs)
# Name the derep-class objects by the sample names
names(derepFs) <- sample.names
names(derepRs) <- sample.names
```

Big Data: The tutorial dataset is small enough to easily load into memory. If your dataset exceeds available RAM, it is preferable to process samples one-by-one in a streaming fashion: see the DADA2 Workflow on Big Data for an example.

Sample Inference

We are now ready to apply the core sample inference algorithm to the dereplicated data.

```
dadaFs <- dada(derepFs, err=errF, multithread=2) # Set mul
```

```
## Sample 1 - 4047 reads in 1248 unique sequences.
## Sample 2 - 4638 reads in 1602 unique sequences.
## Sample 3 - 4473 reads in 1621 unique sequences.
## Sample 4 - 5239 reads in 1499 unique sequences.
## Sample 5 - 4669 reads in 1271 unique sequences.
## Sample 6 - 4755 reads in 1184 unique sequences.
## Sample 7 - 3981 reads in 1371 unique sequences.
## Sample 8 - 4987 reads in 1205 unique sequences.
## Sample 9 - 3709 reads in 1054 unique sequences.
## Sample 10 - 4115 reads in 1139 unique sequences.
## Sample 11 - 4507 reads in 1579 unique sequences.
## Sample 12 - 4626 reads in 1529 unique sequences.
## Sample 13 - 4321 reads in 1474 unique sequences.
## Sample 14 - 5167 reads in 989 unique sequences.
```

Inspect the dada-class object

```
dadaFs[[1]]
```

```
## dada-class: object describing DADA2 denoising results
## 66 sequence variants were inferred from 1248 input uniq
## Key parameters: OMEGA_A = 1e-40, OMEGA_C = 1e-40, BAND_S
```

The getSequences and getUniques functions work on just about any dada2-created object. getUniques returns an integer vector, named by the sequences and valued by their abundances. getSequences just returns the sequences.

```
head(getSequences(dadaFs[[1]]))
```

```
## [1] "TACGGAGGATGCGAGCGTTATCCGGATTTATTGGGTTTAAAGGGTGCGTAC
## [2] "TACGGAGGATGCGAGCGTTATCCGGATTTATTGGGTTTAAAGGGTGCGTAC
## [3] "TACGGAGGATTCAAGCGTTATCCGGATTTATTGGGTTTAAAGGGTGCGTAC
## [4] "TACAGAGGTCTCAAGCGTTGTTCGGAATCACTGGGCGTAAAGCGTGCGTAC
## [5] "TACGGAGGATGCGAGCGTTATCCGGATTTATTGGGTTTAAAGGGTGCGTAC
## [6] "TACGGAGGATGCGAGCGTTATCCGGATTTATTGGGTTTAAAGGGTGCGTAC
```

DADA2 Options: Multithreading

All computation-intensive functions in the dada2 R package have optional multithreading via the `multithread` argument.

- `multithread = FALSE`: No multithreading. The default.
- `multithread = TRUE`: Detect the number of available threads, and use that many. The fastest.
- `multithread = N`: Use N threads. A way to be a good citizen on shared servers and allow processing for other tasks.

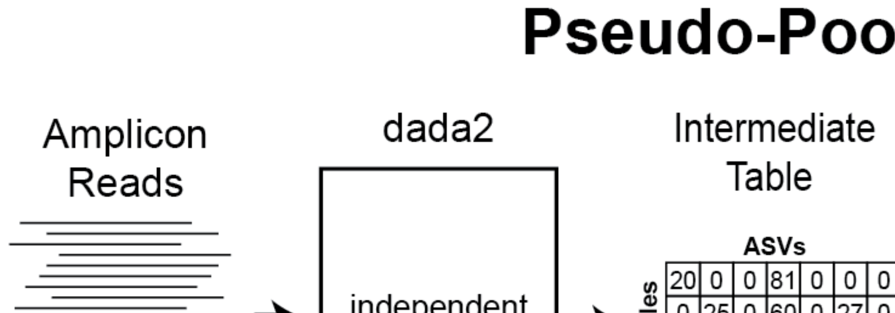
Usually you will want to turn multithreading on!

DADA2 Options: Pooling

Pooling can increase sensitivity to rare per-sample variants. `dada(..., pool=TRUE)`

The cost of pooling is increasing memory and computation time requirements. Pooled sample inference scales quadratically in the number of samples, while the default independent sample inference scales linearly.

Pseudo-pooling approximates pooling in linear time. `dada(..., pool="pseudo")`

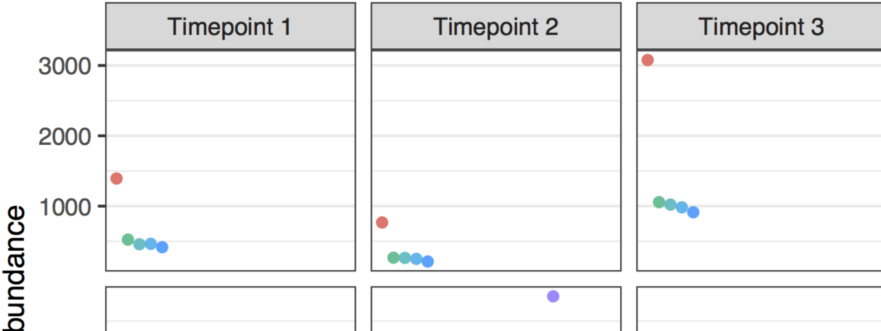


DADA2 Options: Non-Illumnina sequencing technologies

For pyrosequencing data (e.g. **454 or Ion Torrent**) we recommend a slight change in the alignment parameters to better handle those technologies tendency to make homopolymer errors.

```
foo <- dada(..., HOMOPOLYMER_GAP_PENALTY=-1, BAND_SIZE=32)
```

For PacBio CCS amplicon sequencing, see our recent paper for evaluation, and the reproducible workflows from the paper for guidance.



DADA2 Options: Sensitivity

Sensitivity options

- `OMEGA_A`: The key sensititivy parameters, controls the p-value threshold at which to call new ASVs.
- `OMEGA_C`: The error-correction threshold. One alternative is to turn off error-correction.
- `MIN_ABUNDANCE`: Sets a minimum abundance threshold to call new ASVs.
- `MIN_FOLD`: Minimum fold overabundance relative to error model for new ASVs.

See also the `priors` argument to raise sensitivity (at no cost to specificity) for sequences you expect might be present.

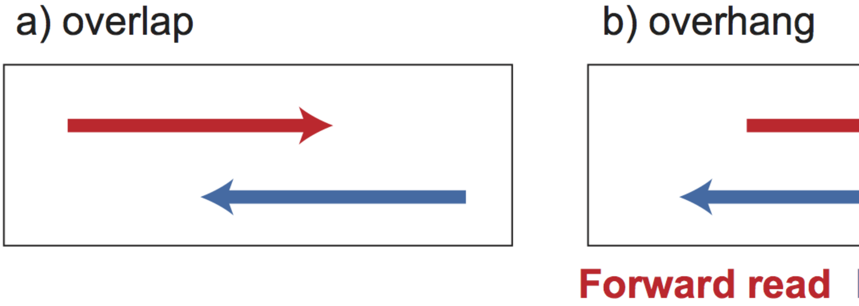
Merge, Table, Remove Chimeras, Sanity Check

Merge Paired Reads

```
mergers <- mergePairs(dadaFs, derepFs, dadaRs, derepRs, ver
```

Most reads should pass the merging step! If that isn't the case, are you sure your truncated reads still overlap sufficiently?

Merging options



- If (a): Use normally.
- If (b or a+b): `mergePairs(..., trimOverhang=TRUE)` *(but you probably should have trimmed away the overhang earlier, see ITS workflow)*
- If (c): `mergePairs(..., justConcatenate=TRUE)`.
- If (a+c or a+b+c): Trouble.

Construct Sequence Table (ASV Table)

```
seqtab <- makeSequenceTable(mergers)
```

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.

```
dim(seqtab)
```

```
## [1]  16 200
```

```
table(nchar(getSequences(seqtab)))
```

```
##
## 252 253 254
##  62 135   3
```

The lengths of the merged sequences all fall in the expected range for this amplicon.

Remove chimeras

Chimeric sequences are identified if they can be exactly reconstructed by combining a left-segment and a right-segment from two more abundant "parent" sequences.

```r
seqtab.nochim <- removeBimeraDenovo(seqtab, method="consens
# Set multithread=TRUE to use all cores
sum(seqtab.nochim)/sum(seqtab)
```

```
## [1] 0.9209602
```

In some cases, most sequences will be chimeric. But most reads should not be. If they are, you probably have unremoved primers.

If you used `pool=TRUE` during sample inference, you should use `method="pooled"` for chimera removal.

Track reads through the pipeline

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, ge
colnames(track) <- c("input", "filtered", "denoisedF", "der
rownames(track) <- sample.names
head(track)
```

```
##              input filtered denoisedF denoisedR merged nor
## 806rcbc288    4497     4047      3918      3949   3632
## 806rcbc289    5131     4638      4480      4563   4191
## 806rcbc290    4921     4473      4315      4354   3972
## 806rcbc291    5886     5239      5096      5165   4773
## 806rcbc292    5116     4669      4550      4607   4313
## 806rcbc293    5318     4755      4584      4695   4379
```

Looks good! We kept the majority of our raw reads, and there is no over-large drop associated with any single step.

SANITY CHECK: Read Tracking

```r
head(track)
```

```
##              input filtered denoisedF denoisedR merged nor
## 806rcbc288  4497     4047      3918      3949   3632
## 806rcbc289  5131     4638      4480      4563   4191
## 806rcbc290  4921     4473      4315      4354   3972
## 806rcbc291  5886     5239      5096      5165   4773
## 806rcbc292  5116     4669      4550      4607   4313
## 806rcbc293  5318     4755      4584      4695   4379
```

- If a majority of reads failed to merge, you may need to revisit `truncLen` to ensure overlap.
- If a majority of reads were removed as chimeric, you may have unremoved primers.

This is the single most important place to inspect your workflow to make sure everything went as expected!

Assign Taxonomy

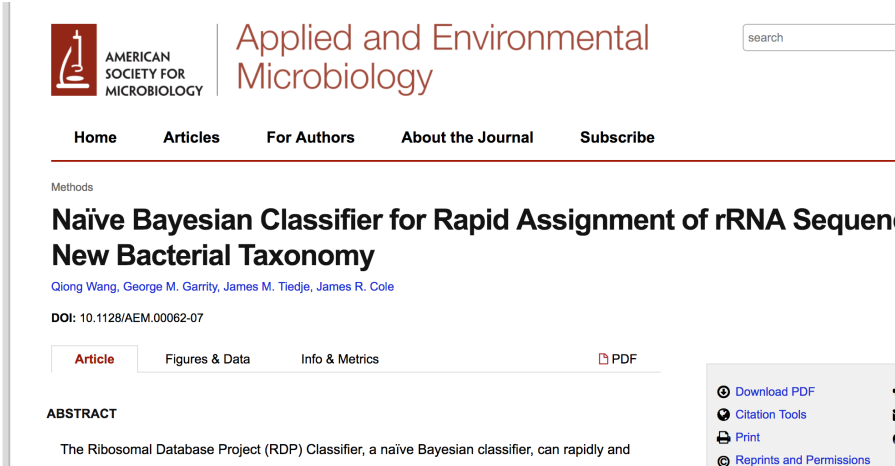
The `assignTaxonomy` function takes as input a set of sequences to ba classified, and a training set of reference sequences with known taxonomy, and outputs taxonomic assignments with at least `minBoot` bootstrap confidence.

```
taxa <- assignTaxonomy(seqtab.nochim, "tax/rdp_train_set_16
```

I recommend the Silva database for 16S data. We are using the RDP database here to keep file sizes down.

Taxonomic assignment methods

The dada2 `assignTaxonomy` function is just a reimplementation of the naive Bayesian classifer developed as part of the RDP project. It is based on shredding reads into kmers, matching against a reference database, and assigning if classification is consistent over subsets of the shredded reads.


AMERICAN SOCIETY FOR MICROBIOLOGY
Applied and Environmental Microbiology
search
Home
Articles
For Authors
About the Journal
Subscribe

Methods

Naïve Bayesian Classifier for Rapid Assignment of rRNA Sequen

New Bacterial Taxonomy

Qiong Wang, George M. Garrity, James M. Tiedje, James R. Cole

DOI: 10.1128/AEM.00062-07

ABSTRACT

The Ribosomal Database Project (RDP) Classifier, a naïve Bayesian classifier, can rapidly and

Taxonomic assignment databases

Having a good reference database is usually **much more important** than the difference between the good taxonomic assignment methods.

What is the reference database for your metabarcoding locus? Is it comprehensive? Appropriate for the environments you are sampling? Do you need to augment or construct your own?

SANITY CHECK: Taxonomic Assignments

```r
head(unname(taxa))
```

```
##      [,1]       [,2]              [,3]                  [,4
## [1,] "Bacteria" "Bacteroidetes"   "Bacteroidia"         "Ba
## [2,] "Bacteria" "Bacteroidetes"   "Bacteroidia"         "Ba
## [3,] "Bacteria" "Bacteroidetes"   "Bacteroidia"         "Ba
## [4,] "Bacteria" "Verrucomicrobia" "Verrucomicrobiae"    "Ve
## [5,] "Bacteria" "Bacteroidetes"   "Bacteroidia"         "Ba
## [6,] "Bacteria" "Firmicutes"      "Clostridia"          "Cl
##      [,5]                  [,6]
## [1,] "Porphyromonadaceae"  NA
## [2,] "Rikenellaceae"       "Alistipes"
## [3,] "Porphyromonadaceae"  NA
## [4,] "Verrucomicrobiaceae" "Akkermansia"
## [5,] "Porphyromonadaceae"  NA
## [6,] "Lachnospiraceae"     NA
```

Do the taxonomies assigned to the top ASVs make sense in the sampled environment?

Handoff to Phyloseq

```r
library("phyloseq"); packageVersion("phyloseq")
```

```
## [1] '1.24.2'
```

Create a phyloseq object from the ASV table and taxonomy assigned by DADA2.

```r
ps <- phyloseq(otu_table(seqtab.nochim, taxa_are_rows=FALSI
               tax_table(taxa))
ps
```

```
## phyloseq-class experiment-level object
## otu_table()   OTU Table:         [ 154 taxa and 16 sampl
## tax_table()   Taxonomy Table:    [ 154 taxa by 6 taxonon
```

Usually you'll want to add sample metadata at this point as well.