Introdution to mare: Microbiota Analysis in R Easily

Katri Korpela

2016-04-13

Introduction

mare is an easy-to-use pipeline for 16S rRNA gene amplicon analysis in the form of an R package. It takes the reads as they come from the sequencing facility, creates taxonomic tables, visualises the results, and finally identifies organisms significantly associated with variables of interest. For OTU clustering and taxonomic annotation the package relies on USEARCH. Although certain options in the pre-processing clearly yield more reliable results than others, the package is flexible and accommodates various choices in the pre-processing and statistical analysis. The package has been tested with Illumina MiSeq reads. This guide takes you through a basic analysis. More information on the different functions can be obtained by typing ?FunctionName in R. I also recommend reading the documentation for USEARCH.

Prerequisites

- R
- Recommended: R studio
- USEARCH version 8 (http://www.drive5.com)
- Optionally a reference database and pre-trained taxonomy confidence files. You may also use one of the reference databases that come with mare. Alternatively, you may use either the RDP database from http://drive5.com/usearch/manual/utax_downloads.html, or custom database from TaxMan (http://www.ibi.vu.nl/programs/taxmanwww/). At http://drive5.com/usearch/manual/utax_downloads.html you will find also the pre-trained taxonomic confidence files (which you need to specify the required confidence level on the taxonomic annotations). Select the confidence file which corresponds to your read length (after read processing). If you just want the best guess for all reads/OTUs, you don't need the confidence files. At Taxman you can select a database specific to your primers. Use the non-redundant version. If you want to identify sequences to the species-level, you should use a database from TaxMan.
- Optionally you may choose to conduct the taxonomic annotation using BLAST, which takes a longer time than USEARCH. In that case, you need to install BLAST+ and download the NCBI taxonomy files.
- Only for visualisation and significance testing: A metadata file (.txt file) containing one column with the sample names and any number of optional columns. The sample names for the taxonomic tables are taken from the beginning of the read file names, ending at "_" (or anything else, which you can specify when running ProcessReads). Make sure your metadata file has a column with these same names, and one row for each sample. In addition to the sample names, the metadata file may contain anything.
- Sequencing reads in one folder. They may be in .fastq.gz or .fastq format.
- Do not include "_" or spaces in the name of the folder or in the name of any other folder in the path!

Short-cut

To get started as easily as possible, you may try the short-cut function called “Simple”. It is an alternative to functions “FormatRefDB”, “ProcessReads”, and “TaxonomicTable”. This function performs the recommended pre-processing protocol, using only the forward reads truncated to 150 bases. Quality-score-based filtering is not conducted, but potential sequencing errors are removing by prevalence-based filtering, discarding unique reads that represent <0.01% of all reads. No OTU clustering is done, but instead the reads are taxonomically annotated, using the confidence cut-off 0, which gives the best guess up to the highest resolution in the reference database. This means that the udb-format database construction is skipped.

This example shows how to run the pre-processing using a reference database from Taxman, restricted to gut-associated taxa, and including only named specied.

```
Simple(forward.reads = list.files(pattern='_R1_'),
       forward.primer = c('CCTACGGGNGGCWGCAG'),
       name.separator = "_",
       folder.name = "",
       usearch.path = '/path/to/usearch8.1.1756_i86osx32',
       refDB = "refDB.fasta",
       gut.specific = T,
       taxman = T,
       named.species = T)
```

Analysis Steps

1. Format your reference data base to be compatible with USEARCH: “FormatRefDB”

This is only needed if you wish to control the confidence level of the taxonomic annotations. If you simply want a best guess for all reads/OTUs, you can use the reference database in the fasta format, don’t need the confidence files, and may skip this step.

The function “FormatRefDB” takes the reference database in fasta format, and the taxonomy confidence file, and creates a new udb-format database compatible with the utax function in USEARCH. If you are analysing human gut communities, you may choose to select the option “gut.specific = TRUE”, which removes all non-human-gut-associated taxa. This step may take a while, but you only need to do it once for each reference database that you use. If you use a database from TaxMan (instead of the RDP-reference database from drive5.com), specify “taxman = TRUE”. If you want to include only named species in your reference database, you can specify “named.species = TRUE”.

Here we use the RDP reference file, and create a gut-specific database. We select the confidence file for readlength 120 because we will trim our reads to a similar length.

```
FormatRefDB(refDB = '/path/to/utaxref/rdp_16s_trainset15/fasta/refdb.fa',
            usearch.path = '/path/to/usearch8.1.1756_i86osx32',
            confidence.file = '/path/to/utaxref/rdp_16s_trainset15/taxconfs/120.tc',
            gut.specific = TRUE)
```

2. Process the reads: “ProcessReads”

The function “ProcessReads” uses R to trim primers and to dereplicate the reads, and calls USEARCH to perform paired-end read merging, truncation to a set length, and quality filtering. All of these steps are optional, and you may pick and choose different options. The function takes a list of your forward read file names and (if you have paired-end reads) a list of your reverse read file names. The function assumes that the names of your read files start with the sample name (corresponding to the sample name column in your metadata file), followed by an underscore. If your sample names are separated from the rest of the read names by something else, you can specify that (option name.separator). This is relevant only if you plan to use the function “Organise” to organise the taxonomic tables to match your metadata file and further analyse the data using information in the metadata file. You may also specify the primer sequences that you would like to remove. You must specify the path to usearch8 on your computer (usearch.path).

There are several options for read processing. If you have paired-end reads, you may choose to merge the forward and reverse reads (“merge = TRUE”). In that case, you also need to specify minimum merged read length (min.merged.length), minimum overlap length (min.overlap), and maximum number of mismatches in the overlap region (max.mismatches).

Instead of merging paired-end reads, you may choose to use only the forward reads, which is the default setting. Using only the reverse reads is not an option, as the forward reads are usually better quality. In this case, you may choose to truncate the forward reads to a specific length (“truncate = TRUE” and trucation.length = the desired readlength). The read quality tends to degrade toward the end, so truncation essentially removes uninformative low-quality bases. Often a clear quality-decline begins at ca. 150b, so this may be a good cut-off length (specify trucation.length = 150).

After optionally merging the paired-end reads, or truncating the forward reads to a specific length, you may choose to quality-filter the reads (“filter = TRUE”). In that case, you must specify the minimum acceptable readlength (min.readlength). The function removes all reads that are shorter than this. You may also alter the default setting for the quality score, at which the read is truncated (trunc.quality = 2 by default), and the maximum expected number of errors per read (max.expected.error = 1 by default). The quality scores may not be 100% reliable, so quality filtering may or may not be useful.

All pre-processed reads are truncated to obtain equallength reads. The final read length must be specified (“readlength = desired read length”). Shorter reads are removed.

The reads are dereplicated, rare reads removed, and chimeras are removed. Rare reads are likely to contain errors: the move often a unique read is observed in the data, the less likely it contains errors. This can therefore be considered an effective quality-filtering step, which does not rely on the quality scores. To specify the the cut-off for rare read removal (as a percentage of the total number of reads in the whole dataset), use min.read.prevalence. By default, reads representing <0.01% of all reads in the total dataset are removed. Finally, OTU clustering is conducted and an OTU table and a read table are created.

As a result, the function will create two new folders, which you need to name by specifying the option “folder.name”. The processed reads will go the new folder and if you wish to try different processing options, you can easily create a new folder for each processing option.

Here we take the forward reads, remove the primer, truncate the reads to 150b, quality-filter, and remove all reads, whose prevalence in the dataset is <1%.

```
ProcessReads(forward.reads=list.files(pattern='_R1_'),
             forward.primer = c('CCTACGGGNGGCWGCAG'),
             usearch.path = '/path/to/usearch8.1.1756_i86osx32',
             truncate = T, trucation.length = 150,
             filter = T, min.readlength = 150,
             readlength = 150,
             min.read.prevalence = 0.01,
             folder.name = 'truncatedfiltered')
```

3. Create taxonomic tables: “TaxonomicTable”

The function “TaxonomicTable” takes the processed reads, assigns taxonomies to the reads or the OTUs, and makes taxonomic tables at different taxonomic levels (phylum to genus, optionally also species). Taxonomic annotation is done by USEARCH.

The taxonomic annotation may be based on the OTUs or the reads themselves (recommended). If you choose to annotate the OTUs, you essentially assume that all reads within the OTU have the same taxonomy. This is probably not the case, as OTU clustering is unlikely to be perfectly accurate. The OTU clustering algorithm used by USEARCH is very good at estimating the true number of species-level taxa in your data, so it is useful for richness analyses and is therefore done even if you choose to annotate the reads.

For taxonomic annotation, you must specify a confidence cut-off: the higher the cut-off the less error in the annotations, but the more unnamed reads/OTUs. If you wish to use a fasta-format database and ignore the confidence level, you must specify “confidence.cutoff = 0”. If you wish to go the species-level, you must use a reference database that has species-level annotations, and specify “confidence.cutoff = 0”.

To tell the program where the processed reads are, you specify folder.name, which must the same as the folder.name given to the ProcessReads function.

Here we use the gut-specific reference database created by FormatRefDB and annotate the reads (not the OTUs), using the taxonomic confidence cut-off 0.5.

```
TaxonomicTable(usearch.path = '/path/to/usearch8.1.1756_i86osx32',
               refDB = 'GutDB.udb',
               annotate.reads = T,
               folder.name = 'truncatedfiltered',
               confidence.cutoff = 0.5)
```

Alternatively to “TaxonomicTable”, you may use the function “Blast” to annotate the OTU/reads. This can be slow. For Blast, you need BLAST+ command line application intallled ftp://ftp.ncbi.nlm.nih.gov/blast/executables/blast+/, and the NCBI taxonomy files ftp://ftp.ncbi.nih.gov/pub/taxonomy/. You also need the R package CHNOSZ.

```
install.package('CHNOSZ') # Do this only the first time you use the Blast function!
Blast(nonchimeric = '/path/to/nonchimeric.fasta',
      read.table = '/path/to/readtable.txt',
      NCBItaxonomy = '/path/to/NCBItaxonomyfiles')
```

Now we are done with the pre-processing and move on to visualise and analyse the data.

4. Copy the taxonomic tables to a new folder: “CopyFiles”

This step is optional. You may use the function “CopyFiles” to create a new folder (specified by the “copy.to” option) and copy the taxonomic tables there. It will also automatically move your current R working directory to the new folder. If you have a metadata file in the same directory where the raw reads are, you can specify the option “meta”, and it will be copied at the same time. The function automatically searches through all folders with the name *TaxonomicTables within the current directory for files with the name* table.txt. If you want to copy only specific files from specific folders, these can be specified by the options “folders” and “files”.

```
CopyFiles(copy.to = '/path/to/analysis/directory',
```

```
    meta = 'meta.txt')
```

5. Organise the taxonomic tables to match your metadata: “Organise”

This step is important to make sure the taxonomic tables and the metadata file are in matching order. Make sure the metadata file has a column with EXACTLY the same sample names at the samples names in the taxomic tables. It may be a good idea to chech at this point. The name of this sample name column can be given to the function by specifying “sample.names”. The function finds automatically the taxonomic tables and the OTU table in the current working directory. You can also specify these file names. In addition to organising the files, the function will calculate the OTU richness (number of OTUs) and diversity (inverse Simpson diversity index, based on the OTU table, using the vegan package) for each sample and save these in the metadata file. The function will also automatically plot the OTU richness against the number of reads per sample, which is a useful diagnostic to evaluate whether some samples may have too few reads to be comparable to the others. In case you have several samples collected at different time points from the same individuals (or otherwise related samples), you can specify “subject.ID” and “time” (numeric variable in the metadata file giving the different sampling time points), and the function will calculate the within-subject similarity (Pearson correlation) and dissimilarity (Bray-Curtis) from one time point to another.

Here we organise the files based on the metadata file called “meta.txt”, in which the sample names are given in the column “sample.name”.

```
Organise(meta = 'meta.txt', sample.names = 'sample_name')
```

6. Visualise the overall microbiota composition using principal coordinates analysis: “PCoA”

“PCoA” performs principal coordinates analysis (using the R-package vegan) on a taxonomic table, using Bray-Curtis dissimilarities. Select which components to show by specifyng “components”; by default the first two will be shown. By default, the function uses the absolute counts of the taxa, but you may choose to use the relative abundances, by specifying “relative = T”. You may remove samples with low read counts (“readcount.cutoff”) and select only certain groups of samples based on a grouping variable (“select.by” specifies the variable and “select” specifies which group to select). Optionally, the function shows group membership (specified by the “group” option) by color and calculates the percentage of variation attributable to group membership (using the adonis function in vegan). Optionally, it also calculates interpolated values for a continuous variable (“background.variable”) in the principal coordinate space using the R-packages gstat and sp, and finally plots the ordination with the background. To save the plot as pdf, specify “pdf = T”.

Here we conduct PCoA using the organised genus-leval data, highlighting the different treatment groups by colour (assuming we have in the metadata file a variable named treatment), and colouring the background based on the number of reads.

```
PCoA(taxonomic.table = 'organised_truncatedfiltered_genus_table.txt',
     meta = 'meta.txt',
     group = 'treatment',
     background.variable = 'ReadCount')
```

7. Find the most abundant and prevalent taxa: “CommonTaxa”

Function “CommonTaxa” is the only function in the package that creates an R object, which needs to be assigned a name. The function finds the most abundant taxa (mean relative abundance cut-off specified by “mean.abundance”) and the most commonly observed taxa (prevalence cut-off specified

by “prevalence”). The function will return a list with two components: “Abundant”, listing the names of the most abundant taxa, and “Prevalent”, listing the names of the most prevalent taxa.

Here we find the genus-level taxa that occur in at least 50% of the samples and whose mean relative abundance in the total dataset is at least 1%. To get a vector of the names of the most abundant and most prevalent taxa, we also unlist them.

```
common<-CommonTaxa(taxonomic.table = 'organised_truncatedfiltered_genus_table.txt',
                   mean.abundance = 0.01,
                   prevalence = 0.5)
abundant <- unlist(common$Abundant)
prevalent <- unlist(common$Prevalent)
```

8. Visualise group differences: “GroupPlot”

Function “GroupPlot” creates several plots which may be helpful in visualising group differences, partly utilising the packages ggplot2 and beanplot. To run it, you need to specify which taxa to include in the plots, and a good starting point is the result from “CommonTaxa”. All of the plots visualise relative abundances of the taxa. Samples with a low number of reads may be discarded (“readcount.cutoff”) and only a certain group may be selected (“select.by” specifies the variable and “select” specifies which group to select). To save the plots as a pdf, specify “pdf = T”. You can select which plots to create by specifying “box/stacked/bean/bar = T/F”.

Here we plot the differences between the different treatment groups in the most abundant taxa, and visualise the relationship between the taxa and the number of reads per sample, separately in the different treatment groups.

```
GroupPlot(taxa = abundant,
          group = 'treatment',
          taxonomic.table = 'organised_truncatedfiltered_genus_table.txt',
          meta = 'meta.txt',
          covariate='ReadCount')
```

9. Visualise associations to a covariate: “CovariatePlot”

Function “CovariatePlot” plots the relative abundance of a selected subset of taxa against a given covariate. By default, it plots the loess smooth, but you may also plot linear associations by specifying “smooth.method = ‘lm’”.Samples with a low number of reads may be discarded (“readcount.cutoff”) and only a certain group may be selected (“select.by” specifies the variable and “select” specifies which group to select). To save the plot as pdf, specify “pdf = T”.

Here we plot the most abundant taxa against overall microbiota richness.

```
CovariatePlot(taxa = abundant,
              taxonomic.table = 'organised_truncatedfiltered_genus_table.txt',
              meta = 'meta.txt',
              covariate='ReadCount')
```

10. Find significant associations: “GroupTest”, “CovariateTest”, “ChangeTest”

There are three functions that can be used to test which bacterial taxa are significantly associated with variables of interest. If you want to test for differences between groups (irrespective of how many

groups you have), use the function “GroupTest”. If you want to test for associations between the taxa and a continuous variable, use the function “CovariateTest”. If you have several time points from the same subjects, and want to test for associations between the change in bacterial taxa and a given grouping variable or a covariate, use “ChangeTest”.

The functions take the taxonomic table, the metadata file, and the grouping variable or covariate of interest. In addition you can specify up to 5 potentially confounding variables. You may also specify the minimum prevalence required for the taxon to be included, thus eliminating potentially spurious results by including rarely-observed taxa. You may also specify a zero-inflation cut-off, which tells the function that taxa occurring less often than the cut-off will be treated as zero-inflated. In “GroupTest” and “CovariateTest”, if you have several samples from the same individuals or otherwise non-independent samples (relatives, individuals from the same geographical area, etc.), you can specify “subject.ID”. The function then assumes that the variable given to “subject.ID” in the metadata file contains information on which samples are related, and will use that variable as the random factor in the model. In that case, the function performs generalized linear mixed models (using the function glmmadmb from the R-package glmmADMB), assuming negative binomial error distributions. If all your samples are independent (no subject.ID specified), the function performs generalized linear models (using the function glm.nb from the package MASS). In both cases, the function controls for the effect of the number of reads per sample by keeping the number of reads as an offset in the model. The function writes a p-value table, which also contains the FDR-corrected p-values, and plots a figure showing the significantly associated bacterial taxa, optionally also saving the plots as pdf-files. Function “ChangeTest” performs linear models (regression or anova).

Here we test for differences between treatment groups, removing all samples with <2000 reads. Values >3 standart deviations from the mean are replaced by the cut-off value to limit the effect of outliers. Taxa occurring in <30% of the samples are ignored.

```
GroupTest(taxonomic.table='organised_truncatedfiltered_genus_table.txt',
          meta = 'meta.txt',
          group = 'treatment',
          readcount.cutoff = 2000,
          outlier.cutoff = 3,
          min.prevalence = 0.3)
```

Here we test for association with the overall microbiota richness, removing all samples with <2000 reads. Values >3 standart deviations from the mean are replaced by the cut-off value to limit the effect of outliers. Taxa occurring in <30% of the samples are ignored.

```
CovariateTest(taxonomic.table='organised_truncatedfiltered_genus_table.txt',
          meta = 'meta.txt',
          covariate = 'richness',
          readcount.cutoff = 2000,
          outlier.cutoff = 3,
          min.prevalence = 0.3)
```