Documentation:
Analyzing microbial communities using high-throughput 16S rRNA sequencing data.

J. Gregory Caporaso[1*], Justin Kuczynski[2*], Jesse Stombaugh[1*], Kyle Bittinger[3], Frederic D. Bushman[3], Elizabeth K. Costello[1], Noah Fierer[4], Antonio Gonzalez Peña[5], Julia K. Goodrich[5], Jeff I. Gordon[6], Gavin Huttley[7], Scott T. Kelley[8], Dan Knights[5], Jeremy E. Koenig[9], Ruth E. Ley[9], Cathy A. Lozupone[1], Daniel McDonald[1], Brian D. Muegge[6], Megan Pirrung[1], Jens Reeder[1], Joel R. Sevinsky[10], Peter J. Turnbaugh[6], Will Van Treuren[1], William A. Walters[2], Jeremy Widmann[1], Tanya Yatsunenko[6], Jesse Zaneveld[2] and Rob Knight[1,11**]

1. Department of Chemistry and Biochemistry, UCB 215, University of Colorado, Boulder, CO 80309
2. Department of Molecular, Cellular and Developmental Biology, UCB 347, University of Colorado, Boulder, CO 80309
3. Department of Microbiology, Johnson Pavilion 425, University of Pennsylvania, Philadelphia, PA 19104
4. Cooperative Institute for Research in Environmental Sciences, University of Colorado, Boulder, CO 80309, USA.; Department of Ecology and Evolutionary Biology, University of Colorado, Boulder, CO 80309, USA.
5. Department of Computer Science, University of Colorado, Boulder, Colorado, USA.
6. Center for Genome Sciences, Washington University School of Medicine, St. Louis, MO 63108
7. Computational Genomics Laboratory, John Curtin School of Medical Research, The Australian National University, Canberra, Australian Capital Territory, Australia.
8. Department of Biology, San Diego State University, San Diego CA 92182
9. Department of Microbiology, Cornell University, Ithaca NY 14853
10. Luca Technologies, 500 Corporate Circle, Suite C, Golden, Colorado 80401
11. Howard Hughes Medical Institute

1. Introduction

1.1. Who Should Use this Software?

This software is designed for scientists investigating the microbial diversity within and between samples using SSU (16S or 18S) rRNA gene sequence surveys where the gene sequences are generated by PCR and (mostly) sequenced with 454 pyrosequencing. In order to analyze the diversity of sequences within multiple samples with a single pyrosequencer run, barcodes are assigned to samples, and the amplicons are pooled for sequencing. This software recognizes the barcodes and sorts sequences back to their original samples, then applies a comprehensive set of analysis tools to assess the diversity within samples and compare samples to one another. The analysis also incorporates data provided by the user, to relate known properties of samples or sets of samples to their microbial constituents.

This software is designed for use by scientists familiar with a basic approach to the analysis of microbial diversity. Familiarity with the concepts of OTUs, phylogenetic trees, diversity estimates, and methods of community comparisons such as UniFrac and PCoA, are necessary for the interpretation of the results. The software is designed for wide use, is user-friendly, and does not require specialized skills. it is designed for students and professionals, or anyone wishing to analyze a microbial diversity dataset.

1.2. Getting the Software

The QIIME software can be found on Sourceforge at the following location: http://qiime.sourceforge.net/

1.3. Microbial Community Analysis

This software is designed to generate lists of OTUs from very large datasets (mostly those generated by pyrosequencing), and to perform phylogenetic and nonphylogenetic OTU-based analyses. It performs sequence quality checks and chimera-checking, chooses OTUs, performs Sequence-, UniFrac- and OTU-based clustering of samples from which the sequences were obtained, assigns sequences to microbial lineages, displays heatmaps of common and rare OTUs in samples, performs rarefaction analysis for an assessment of depth of coverage, builds very large trees for tree-based community comparisons such as UniFrac, and applies a network-based analysis.

1.4. Sequencing Technologies

Software is in principle compatible with all platforms: 454, Illumina, and/or Sanger sequencing. Different technologies require different strategies for building the tree: for short reads or for heterogeneous reads that don't overlap, it is more effective to insert the reads into a tree built using full-length sequences. This is because trees built with reads of less than 200 nucleotides are often highly inaccurate (though still often sufficient for tree-based community comparisons such as UniFrac), and trees generally cannot be built de novo with sequences that do not overlap at all. For combining heterogeneous datasets, the best approach is usually to use ***split_libraries.py*** on each dataset separately, combine the FASTA files, make an OTU table using BLAST (rather than cd-hit), and then use the combined OTU table and a combined sample mapping file for downstream analysis. Another important consideration is error rate: pyrosequencing reads that are not denoised (see (Quince et al., 2009)) can lead to very large numbers of artifactual OTUs, even at relatively high levels. However, this noise problem affects alpha diversity estimates (estimates of diversity within a sample) far more than beta diversity estimates (diversity across samples, e.g. clustering with UniFrac). See (Reeder & Knight, 2009) for a commentary on this issue.

2. Input files

2.1. What Files Do I Need?

1. 454-machine generated FASTA file(s) (.fna or .fasta)
2. Quality score file(s) (.qual) - Optional
3. sff file(s) (.sff) - Optional
4. A mapping file generated by the user. This file contains all of the information about the sequences necessary to perform the data analysis: the name of each sample, the barcode assignment of each sample (if preprocessing step is activated), the linker and primer sequence used to amplify each sample, any metadata that relates to the samples (for instance, health status or sampling site), and any additional information relating to specific samples that may be useful to have at hand when considering outliers (for example, what medications a patient was taking

at time of sampling). See format specifications below (**Section 2.2.4**).

2.2. File Format Details

These are general guidelines that apply to multiple files:

1. Files should have proper file type suffix: E.g. '.fna' or '.fasta' for FASTA files, '.qual' for quality score files, '.sff' for flowgram files, '.txt' for the mapping files.
2. Do not use spaces in the filename. Use underscores or MixedCase instead. For example: 'amazon soil.fna' is not allowed, should be 'amazon_soil.fna' or 'AmazonSoil.fna'

2.2.1 FASTA File (.fna)

This file can be generated from a 454 sff file using the command:

sffinfo -s NAME_OF_SFF_FILES > OUTPUT_FILE.fna

This is an example showing the first few raw sequences in an .fna file:

```
>FV5UW9Z02IMIZQ length=42 xy=3419_3860 region=2 run=R_2009_05_20_11_44_59_
GACATCGGCTATCATGCTGCCTCCCTGNTGNNNNNCANNGCG
>FV5UW9Z02GOSNH length=243 xy=2625_2683 region=2 run=R_2009_05_20_11_44_59_
GACTAACGTCACCATGCTGCCTCCCGTAGGAGTCTGGACCGTGTCTCAGTTCCAGTGTGGCCGTTCATCCTCTCAGACCGGCTACTGAT
CGTCGCCTTGGTGAGCCGTTACCTCACCAACTAGCTAATCAGACGCGGGTCCATCTCATACCACCGGAGTTTTTCACACTGTGCCATGC
AGCACTGTGCGCTTATGCGGTATTAGCAGTCATTTCTAACTGTTATCCCCTGTATGAGGCAGGTT
>FV5UW9Z02FREEG length=271 xy=2245_1046 region=2 run=R_2009_05_20_11_44_59_
GACGAGTCAGTCCATGCTGCCTCCCGTAGGAGTTTGGGCCGTGTCTCAGTCCCAATGTGGCCGGTCACCCTCTCAGGTCGGCTACTGAT
CGTCGCCTTGGTAGGCCGTTACCCCACCAACTAGCTAATCAGACGCGGGTCCATCTCATACCACCGGAGTTTTTCACACCGTACCATGC
GGTACTGTGCGCTTATGCGGTATTAGCAGCCGTTTCCAACTGTTATCCCCCTGTATGAGGCAGGTTACCCACGCGTTACTCACCCGTCC
GCCG
...
```

2.2.2 . Quality Score File (.qual) - Optional

This is the 454-machine generated quality score file, which contains a score for each base in each sequence included in the FASTA file. It can be generated from an sff file using the command:

sffinfo -q NAME_OF_SFF_FILES > OUTPUT_FILE.qual

The quality score file that corresponds to the above .fna example looks like this:

```
>FV5UW9Z02IMIZQ length=42 xy=3419_3860 region=2 run=R_2009_05_20_11_44_59_
32 32 32 32 32 25 25 25 32 32 32 32 32 32 32 32 32 32 37 40 37 37 30 30 25 25 25 0 25 30
0 0 0 0 0 27 24 0 0 27 27 15
>FV5UW9Z02GOSNH length=243 xy=2625_2683 region=2 run=R_2009_05_20_11_44_59_
37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 39 39 39 39 36 29 29 29 29 29 29 29
33 33 33 39 39 39 39 40 40 40 40 40 40 40 40 37 37 37 37 38 38 36 36 36 37 37 37 37 37 37
37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 33 33 33 33 37 37 37 37 37 37 37 37 37
37 37 37 37 38 38 38 38 37 37 37 37 37 37 37 37 37 37 37 36 36 36 37 36 28 28 28 28 28 36
37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 36 36 36 36 36 37 37 37 37 37 37 37 36 36 36
36 37 37 37 37 37 32 22 22 22 22 22 22 33 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37
37 37 37 37 37 37 37 37 37 36 36 36 36 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 36
28 28 28 33 33 35 31 37 37 35 31 31 26 19 19 22 22 22 26 36 36 36 36 36 36 36 36 36 36 36
36 25 25
>FV5UW9Z02FREEG length=271 xy=2245_1046 region=2 run=R_2009_05_20_11_44_59_
37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 40 40 40 40 40 40 40 40 40 40 40 40
40 40 40 40 40 38 38 38 40 40 40 40 40 40 40 37 37 37 37 37 40 40 40 37 37 37 37 37 37 37
37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37
37 37 37 37 33 33 34 34 37 34 33 34 38 38 37 37 37 37 37 37 37 37 33 33 33 33 33 37 37 37
37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37
37 37 37 37 37 37 37 37 40 40 40 40 40 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37
37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37
37 37 37 37 37 37 38 38 38 40 38 38 38 38 33 33 33 33 33 40 38 38 38 38 38 38 38 38 40 38
```

```
38 38 38 38 40 32 32 32 32 32 32 32 32 32 32 32 32 31 31 32 25 25 25 31 30 21 20 20 25 28
30
...
```

2.2.3 Flowgram File (.sff) - Optional

This file is found in the 454 data analysis folder in the "sff" directory. It contains all analyzed data from the sequencing run. This file and the qual file, are optional, which to allows for non-454 data (for example, using Sanger reads you may have already performed quality filtering on by the time you started using the pipeline, and/or sequences obtained from GenBank, which may not have quality scores. In these cases, the different kinds of data need to be processed separately with ***split_libraries.py***, and then the resulting FASTA files can be combined.

2.2.4 Mapping File (tab-delimited .txt)

The mapping file is generated by the user. This file contains all of the information about the sequences necessary to perform the data analysis: the name of each sample (SampleID), the DNA barcode used for each sample (BarcodeSequence), the linker and primer sequence used to amplify a given sample (LinkerPrimerSequence), any metadata that relates to the samples (for instance, health status or sampling site), and any additional information relating to specific samples that may be useful to have at hand when considering outliers (for example, what medications a patient was taking at time of sampling). See format specifications below.

The first few lines of a mapping file must look like the following:

```
#SampleID   BarcodeSequence   LinkerPrimerSequence   HostMood  Description
#Example mapping file for QIIME.  Notes may be added on this line
sample1     AGCACGAGCCTA      YATGCTGCCTCCCGTAGGAGT  tired     ear_sample_from_subject228
...
```

You are strongly encouraged to validate your mapping file using ***check_id_map.py*** before attempting to analyze your data. This tool will check for errors and make suggestions for other aspects of the file to be edited. Errors and warnings will be written to a log file and suggested corrections will be written to a '_corrected.txt' version of the mapping file, both located in the specified output directory. Currently, the number of valid characters in the headers and data of the mapping file are quite restricted to ensure compatibility with downstream programs. For headers, only alphanumeric and underscores are accepted. Sample IDs, descriptions, and other metadata additionally accept the following characters: .+-%. Barcodes and linker/primer sequences must be IUPAC DNA characters. If invalid characters are found, the '_corrected.txt' output file will have each of these replaced with an underscore.

The mapping file relates barcodes in the FASTA file to each sample and their related metadata. Each FASTA file must have at least one mapping file but multiple mapping files can be defined for any given FASTA file. For example, if you have bundled several unrelated studies into one 454 run (for instance, a mouse study, a soil study and a fish study), and need to analyze each study separately, you would generate three separate mapping files that specify a subset of samples and their associated metadata. Alternatively, you can combine multiple runs (e.g. multiple 454 runs, multiple FASTA files) with a single mapping file.

The mapping file is organized to contain data in columns with headers. As such, it is usually generated by users in Excel. However, it cannot be submitted in Excel format, it must be submitted in tab-delimited text format. Simply saving an excel spreadsheet in tab-delimited text format may be problematic because it will include extra trailing empty columns. One way around this is to copy and paste the highlighted data into a more simple text editing program.

The mapping file format is tab-delimited text. This must be plain text, therefore Word and Excel documents cannot be read directly. The first line of the mapping file is the header line and it MUST start with a "#" character. Optionally, lines immediately following the header line that start with a "#" character will be treated as run description lines. This section is for descriptions that apply to all samples in the mapping file (as opposed to / in addition to the descriptions that apply to each individual sample).

Each column header MUST contain alphanumeric (a-z, A-Z and 1-9) and/or underscore ("_") characters only, where the header MUST start with letter. All other characters (e.g. $, *, ^, etc) are not supported at this time and use of those characters may cause problems downstream in the QIIME pipeline. Sample IDs, descriptions, and metadata fields additionally allow the use of plus(+), minus(-), period(.), and percent (%) characters.

Currently, the user has the ability to define their own column headers, however; QIIME will be adopting the MIENS

standard, therefore all column headings MUST correspond the proper MIENS nomenclature (http://gensc.org/gc_wiki/index.php/MIENS). The following details the current mapping file guidelines:

1. The first column header must be "#SampleID", and the data in this column must contain unique (short and meaningful) sample identifiers containing only alphanumeric, period (".") and/or underscore ("_") characters.
2. The second column header must be "BarcodeSequence", where each value in that column corresponds to the barcode used for each sample. Only IUPAC DNA characters are acceptable.
3. The third column header must be “LinkerPrimerSequence”, where each value in that column corresponds to the primer used to amplify that sample. Only IUPAC DNA characters are acceptable.
4. All subsequent column headers (except the last one) are metadata headers. For example, a "Smoker" column would include either "Yes" or "No". Note that the data in each column is assumed to be categorical unless specified otherwise. Categorical data columns must include at least 2 unique values per column. All metadata must be composed of only alphanumeric, underscore ("_"), period ("."), minus sign ("-"), plus sign ("+") and/or percentage ("%") characters. For missing data, write "NA", do not leave blanks.
5. The last column of the mapping file must be named "Description". Information in this column includes information that is unique to each sample, such as the medications taken by the patient, or any other descriptive information. There are no formatting restrictions for this column. Sample/Run Description should be kept brief, if possible. Information that applies to all samples in a mapping file should go in the run description section, which is defined as lines starting with a "#" character, immediately following the header line (See example format below.) Information that is specific to a particular sample should go in the "Description" column.

Notes for more than one mapping or FASTA file at once:

1. If you want to use a single mapping file with multiple FASTA files, MAKE SURE NO BARCODES ARE REPEATED BETWEEN THE RUNS.
2. If you are providing multiple mapping files for a single FASTA file, make sure that use of column headers is consistent across all files. (Case-sensitive!) E.g. "Barcode" is not the same as "BarCode" is not the same as "barcode" is not the same as "Bar_code".

Example Mapping Files:

1. Mapping file with a unique "Description" column:

```
#SampleID BarcodeSequence  LinkerPrimerSequence  AnimalDiet    ... Description
A         CGAGTCTAGTTG     YATGCTGCCTCCCGTAGGAGT Carnivore     ... Lion_11
B         CCGACTGAGATG     YATGCTGCCTCCCGTAGGAGT Herbivore     ... Kangaroo_3
C         TCGCCTGAGATG     YATGCTGCCTCCCGTAGGAGT Omnivore      ... Human_34
...
```

2. Mapping file with "run description" lines following the "header line":

```
#SampleID     BarcodeSequence LinkerPrimerSequence AnimalDiet  ... Description
# This is a "run description" example. It can span as many lines as needed
# before the rest of the mapping file starts. Each line must begin with
# the pound '#' character. You do not have to wrap your lines but it may make
# your "run description" section easier to read.
# The run description should contain a general description of the run
# and details that might be useful in interpreting the results such as:
# 1) what type of samples are in this run, 2) where and when they were collected,
# 3) how they were prepped, 4) what type of barcodes and primers were used
# 5) where they were sequenced (e.g. facility) or collected from
# (e.g. public databases), 6) what you are expecting/hoping to find in the run
# 7) any possible problems (technical e.g. during prep/sequencing, experimental
# etc) 8) etc.
# The description in this section will be applied to any individual samples
# that are missing a value in the 'Description' column
#SampleID BarcodeSequence  LinkerPrimerSequence  AnimalDiet    ... Description
A         CGAGTCTAGTTG     YATGCTGCCTCCCGTAGGAGT Carnivore     ... Lion_11
B         CCGACTGAGATG     YATGCTGCCTCCCGTAGGAGT Herbivore     ... Kangaroo_3
C         TCGCCTGAGATG     YATGCTGCCTCCCGTAGGAGT Omnivore      ... Human_34
...
```

3. QIIME Analyses and Parameters

3.1 General Notes

All QIIME analyses are performed using python (.py) scripts, which are located in the qiime directory. To access QIIME python scripts, it may be useful to set an environment variable to the location of the innermost QIIME directory (the one containing ***check_id_map.py***, for example):

```
$ qdir=/path/to/QIIME/
```

Further commands of the form "python QIIME_script.py -o option" can be invoked as "python $qdir/QIIME_script.py -o option". For all path description throughout the Documentation, the "/path/to/" refers to the physical location of each program on your local computer. For instance, the "/path/to/QIIME/" on my computer refers to "/Users/Jesse/Qiime/" on Mac OS X version 10.5.

The user can obtain help about the arguments which can be passed to each script, as well as examples and usage notes, by typing the following in the bash shell:

```
$ python $qdir/script_of_interest.py –h
```

3.2 Sequence Quality Filtering and Library Splitting

3.2.1 Check Mapping File for Errors/Warnings

QIIME script: ***check_id_map.py***

The check id map step is necessary in order to confirm that the user generated mapping file (described above) is properly formatted. For example, the "#SampleID" column is one of the required fields in the mapping file and all entries in this column must be unique. Also, an error will be returned when running ***check_id_map.py*** if there are duplicate entries in the "#SampleID" column. By default, no output will go to stdout or stderr. If verbose is enabled (-v option), a message indicating whether or not errors/warnings were found will go to stdout.

Usage: ***check_id_map.py*** [options]

Input Arguments:

-m MAP_FNAME, --map=MAP_FNAME [REQUIRED]

- This is the user-generated mapping file.

-o OUTPUT_DIR, --output_dir=OUTPUT_DIR [REQUIRED]

- Required output directory for mapping file with corrected characters and log file (by default, invalid characters will be converted to underscores)

-b HAS_BARCODES, --is-barcoded=HAS_BARCODES [Default: 1]

- If the mapping file contains the barcode sequence ("BarcodeSequence"), then the user should set this value as 1 (True), else as 0 (False).

-v, --verbose [Default: False]

- Enable verbose output.

Output:

This script creates two files in the specified output directory: a log file and a suggested corrected mapping file. A log file generated by this module will contain a list of errors, generally header problems, and warnings (e.g., duplicate descriptions, invalid characters in metadata, etc.), along with a location of the error (in row, column format) in the case of

non-header problems. Header problems should be corrected before proceeding. If no invalid characters are found, the corrected file will contain a comment denoting this result.

Examples:

The following is an example of running ***check_map_id.py*** given a user-defined "Mapping_File.txt", which contains barcodes, with the output files going to the output_mapping directory (which will be created in the current directory if it is not present):

```
$ python $qdir/check_id_map.py -m Mapping_File.txt -b 1 -o output_mapping/
```

or using the default "-b" option, which is 1 (True):

```
$ python $qdir/check_id_map.py -m Mapping_File.txt -o output_mapping/
```

If the mapping file does not contain barcodes, you can type the following command:

```
$ python $qdir/check_id_map.py -m Mapping_File.txt -b 0 -o output_mapping/
```

3.2.2 Relabel Sequences According to Sample and Eliminating Low Quality Sequences

QIIME script: ***split_libraries.py***

Since newer sequencing technologies provide many reads per run (e.g. the 454 GS FLX Titanium series can produce 400-600 million base pairs with 400-500 base pair read lengths) researchers are now finding it useful to combine multiple samples into a single 454 run. This multiplexing is achieved through the application of a pyrosequencing-tailored nucleotide barcode design (described in (Parameswaran et al., 2007)). By assigning individual, unique sample specific barcodes, multiple sequencing runs may be performed in parallel and the resulting reads can later be binned according to sample. The script ***split_libraries.py*** performs this task, in addition to several quality filtering steps including user defined cut-offs for: sequence lengths; end-trimming; minimum quality score. To summarize, by using the fasta, mapping, and quality files, the program ***split_libraries.py*** will parse sequences that meet user defined quality thresholds and then rename each read with the appropriate Sample ID, thus formatting the sequence data for downstream analysis. If a combination of different sequencing technologies are used in any particular study, ***split_libraries.py*** can be used to perform the quality-filtering for each library individually and the output may then be combined (described in **Section 1.4**). Sequences from samples that are not found in the mapping file (no corresponding barcode) and sequences without the correct primer sequence will be excluded. Additional scripts can be used to exclude sequences that match a given reference sequence (e.g. the human genome; ***exclude_seqs_by_blast.py***) and/or sequences that are flagged as chimeras (***identify_chimeric_seqs.py***).

Usage: ***split_libraries.py*** [options]

Input Arguments:

-m MAP_FNAME, --map=MAP_FNAME [REQUIRED]

- This is the user-generated mapping file.

-f FASTA_FNAMES, --fasta=FASTA_FNAMES [REQUIRED]

- This is the FASTA file(s) generated from the sequencing process. If more than one FASTA file is generated, all files can be parsed by comma (",") separating each file on the command line.

-q QUAL_FNAMES, --qual=QUAL_FNAMES [Default: None]

- This is the QUAL file(s) generated from the sequencing process. If more than one QUAL file is generated, all files can be parsed by comma (",") separating each file on the command line.

-l MIN_SEQ_LEN, --min-seq-length=MIN_SEQ_LEN [Default: 200]

- This is the minimum sequence length of a sequence to keep in the output FASTA file.

-L MAX_SEQ_LEN, --max-seq-length=MAX_SEQ_LEN [Default: 1000]

- This is the maximum sequence length of a sequence to keep in the output FASTA file.

-t, --trim-seq-length

- If this option is used, then the script will calculate sequence lengths after trimming primers and barcodes.

-s MIN_QUAL_SCORE, --min-qual-score=MIN_QUAL_SCORE [Default: 25]

- This is the minimum average quality score allowed in read to be kept in the output FASTA file.

-k, --keep-primer

- If this option is used, the primers will be kept in the output FASTA file.

-a MAX_AMBIG, --max-ambig=MAX_AMBIG [Default: 0]

- This is the maximum number of ambiguous bases for the sequence to be kept in the output FASTA file.

-H MAX_HOMOPOLYMER, --max-homopolymer=MAX_HOMOPOLYMER [Default: 6]

- This is the maximum length of a homopolymer run to be kept in the output FASTA file.

-M MAX_PRIMER_MM, --max-primer-mismatch=MAX_PRIMER_MM [Default: 0]

- This is the maximum number of primer mismatches allowed for the sequence to be kept in the output FASTA file.

-b BARCODE_TYPE, --barcode-type=BARCODE_TYPE [Default: golay_12]

- This is the type of barcoding used for the experiment, e.g. 4 or hamming_8 or golay_12.

-o DIR_PREFIX, --dir-prefix=DIR_PREFIX [default: .]

- This is the directory location, where all output files should be written.

-e MAX_BC_ERRORS, --max-barcode-errors=MAX_BC_ERRORS [Default: 1.5]

- This is the maximum number of errors in barcode allowed for a sequence to be kept in the output FASTA file.

-n START_INDEX, --start-numbering-at=START_INDEX

- This is the seq id to use for the first sequence to be parsed by this script.

-r, --remove_unassigned

- If this option is used, then all sequences which remain Unassigned are removed from the output FASTA file, otherwise the Unassigned sequences will be kept in the output FASTA file.

-B, --keep-barcode

- This keeps the barcode on the sequence.

-c, --disable_bc_correction [Default: False]

- This disables the attempt to find nearest corrected barcode, which can improve performance.

-w QUAL_SCORE_WINDOW, --qual_score_window=QUAL_SCORE_WINDOW [Default: 0]

- This enables a sliding window test of quality scores. If the average score of a continuous set of w nucleotides falls below the threshold (see -s for default), the sequence is discarded. A good value would be 50. 0 (a value of zero means no filtering). One must pass a qual file (see "-q "parameter) if this functionality is enabled.

Output:

Three files are generated by ***split_libraries.py***:

1. .fna file (e.g. seqs.fna) - This is a FASTA file containing all sequences which meet the user-defined parameters, where each sequence identifier now contains its corresponding sample id from mapping file.
2. histograms.txt - This contains the counts of sequences with a particular length.
3. split_library_log.txt - This file contains a summary of the ***split_libraries.py*** analysis. Specifically, this file includes information regarding the number of sequences that pass quality control (number of seqs written) and how these are distributed across the different samples which, through the use of bar-coding technology, would have been pooled into a single 454 run. The number of sequences that pass quality control will depend on length restrictions, number of ambiguous bases, max homopolymer runs, barcode check, etc. All of these parameters are summarized in this file. If raw sequences do not meet the specified quality thresholds they will be omitted from downstream analysis. Since we never see a perfect 454 sequencing run, the number of sequences written should always be less than the number of raw sequences. The number of sequences that are retained for analysis will depend on the quality of the 454 run itself in addition to the default data filtering thresholds in the ***split_libraries.py*** script. The default parameters (minimum quality score = 25, minimum/maximum length = 200/1000, no ambiguous bases allowed, no mismatches allowed in primer sequence) can be adjusted to meet the user's needs.

Examples:

Using a single 454 run, which contains a single FASTA, QUAL, and mapping file while using default parameters and outputting the data into the Directory "Split_Library_Output".

```
$ python $qdir/split_libraries.py -m Mapping_File.txt -f 1.TCA.454Reads.fna -q 1.TCA.454Reads.qual -o
        Split_Library_Output/
```

For the case where there are multiple FASTA and QUAL files, the user can run the following command as long as there are not duplicate barcodes listed in the mapping file:

```
$ python $qdir/split_libraries.py -m Mapping_File.txt -f 1.TCA.454Reads.fna,2.TCA.454Reads.fna -q
        1.TCA.454Reads.qual,2.TCA.454Reads.fna -o Split_Library_Output/
```

Duplicate Barcode Example:

An example of this situation would be a study with 1200 samples. You wish to have 400 samples per run, so you split the analysis into three runs with and reuse barcoded primers (you only have 600). After initial analysis you determine a small subset is underrepresented (<500 sequences per samples) and you boost the number of sequences per sample for this subset by running a fourth run. Since the same sample IDs are in more than one run, it is likely that some sequences will be assigned the same unique identifier by ***split_libraries.py*** when it is run separately on the four different runs, each with their own barcode file. This will cause a problem in file concatenation of the four different runs into a single large file. To avoid this, you can use the '-s' parameter which defines a start index for ***split_libraries.py***. From experience, most FLX runs (when combining both files for a single plate) will have 350,000 to 650,000 sequences. Thus, if Run 1 for ***split_libraries.py*** uses '-n 1000000', Run 2 uses '-n 2000000', etc., then you are guaranteed to have unique identifiers after concatenating the results of multiple FLX runs. With newer technologies you will just need to make sure that your start index spacing is greater than the potential number of sequences.

To run ***split_libraries.py***, you will need two or more (depending on the number of times the barcodes were reused) separate mapping files (one for each Run, for example one Run1 and another one for Run2), then you can run ***split_libraries.py*** using the FASTA and mapping file for Run1 and FASTA and mapping file for Run2. Once you have independently run split libraries on each file independently, you can concatenate (cat) the sequence files generated. You

can also concatenate the mapping files, since the barcodes are not necessary for downstream analyses, unless the same sample id's are found in multiple mapping files.

Run ***split_libraries.py*** on Run 1:

```
$ python $qdir/split_libraries.py -m Mapping_File.txt -f 1.TCA.454Reads.fna -q 1.TCA.454Reads.qual -o
    Split_Library_Run1_Output/ -n 1000000
```

Run ***split_libraries.py*** on Run 2:

```
$ python $qdir/split_libraries.py -m Mapping_File.txt -f 2.TCA.454Reads.fna -q 2.TCA.454Reads.qual -o
    Split_Library_Run2_Output/ -n 2000000
```

Concatenate the resulting FASTA files for use in downstream analyses:

```
$ cat Split_Library_Run1_Output/seqs.fna Split_Library_Run2_Output/seqs.fna > Combined_seqs.fna
```

Suppress "Unassigned" Sequences Example:

Users may want to only output sequences which have been assigned to a particular sample. To suppress the outputting of "Unassigned sequences", the user can pass the "-r" option, without any additional values.

```
$ python $qdir/split_libraries.py -m Mapping_File.txt -f 1.TCA.454Reads.fna -q 1.TCA.454Reads.qual -o
    Split_Library_Output/ -r
```

Barcode Decoding Example:

The standard barcode types supported by ***split_libraries.py*** are golay (Length: 12 NTs) and hamming (Length: 8 NTs). For situations where the barcodes are of a different length than golay and hamming, the user can define a generic barcode type "-b" as an integer, where the integer is the length of the barcode used in the study.

For the case where the hamming_8 barcodes were used, you can use the following command:

```
$ python $qdir/split_libraries.py -m Mapping_File.txt -f 1.TCA.454Reads.fna -q 1.TCA.454Reads.qual -o
    Split_Library_Output/ -b hamming_8
```

In the case where the barcodes used were different than the golay or hamming, one can define the length of barcode used (e.g. length of 6 NTs), as shown by the following command:

```
$ python $qdir/split_libraries.py -m Mapping_File.txt -f 1.TCA.454Reads.fna -q 1.TCA.454Reads.qual -o
    Split_Library_Output/ -b 6
```

Note: When analyzing large datasets (>100,000 seqs), users may want to use a generic barcode type, even for length 8 and 12 NTs, since the golay and hamming decoding processes can be computationally intensive, which causes the script to run slow. Barcode correction can be disabled with the -c option if desired.

Linkers and Primers:

The linker and primer sequence (or all the degenerate possibilities) are associated with each barcode from the mapping file. If a barcode cannot be identified, all the possible primers in the mapping file are tested to find a matching sequence. Using truncated forms of the same primer can lead to unexpected results for rare circumstances where the barcode cannot be identified and the sequence following the barcode matches multiple primers.

3.3. Sequence Dereplication (i.e. Picking OTUs and Representative Sets)

3.3.1 OTU Picking

QIIME script: ***pick_otus.py***

The OTU picking step assigns similar sequences to operational taxonomic units, or OTUs, by clustering sequences based

on a user-defined similarity threshold. Sequences which are similar at or above the threshold level are taken to represent the presence of a taxonomic unit (e.g., a genus, when the similarity threshold is set at 0.94) in the sequence collection.

Currently, the following clustering methods have been implemented in QIIME:

1. cd-hit (Li & Godzik, 2006; Li, Jaroszewski, & Godzik, 2001), which applies a "longest-sequence-first list removal algorithm" to cluster sequences.
2. blast (Altschul, Gish, Miller, Myers, & Lipman, 1990), which compares and clusters each sequence against a reference database of sequences.
3. Mothur (Schloss et al., 2009), which requires an input file of aligned sequences. The input file of aligned sequences may be generated from an input file like the one described below by running ***align_seqs.py***. For the Mothur method, the clustering algorithm may be specified as nearest-neighbor, furthest-neighbor, or average-neighbor. The default algorithm is furthest-neighbor.
4. prefix/suffix [Qiime team, unpublished], which will collapse sequences which are identical in their first and/or last bases (i.e., their prefix and/or suffix). The prefix and suffix lengths are provided by the user and default to 50 each.
5. Trie [Qiime team, unpublished], which collapsing identical sequences and sequences which are subsequences of other sequences.

Usage: ***pick_otus.py*** [options]

Input Arguments:

-i INPUT_SEQS_FILEPATH, --input_seqs_filepath=INPUT_SEQS_FILEPATH [REQUIRED]

- Path to FASTA file containing sequences, where the sequence ids may contain the sample ids (i.e., resulting FASTA file from ***split_libraries.py***)

-m OTU_PICKING_METHOD, --otu_picking_method=OTU_PICKING_METHOD [Default: cdhit]

- This is the method that should be used for picking OTUs. Valid choices are: cdhit, blast, prefix_suffix, and mothur. The mothur method requires an input file of aligned sequences. Whichever method is chosen, the appropriate 3rd party application must be installed properly.

-c CLUSTERING_ALGORITHM, --clustering_algorithm=CLUSTERING_ALGORITHM [Default: furthest]

- This is the clustering algorithm for the mothur otu picking method. Valid choices are: nearest, furthest, and average.

-M MAX_CDHIT_MEMORY, --max_cdhit_memory=MAX_CDHIT_MEMORY [Default: 400]

- This is the maximum available memory allowed to be used by cdhit (cd-hit's -M) (Mbyte). This option only works with the cdhit method.

-o OUTPUT_DIR, --output_dir=OUTPUT_DIR [Default: ./<OTU_METHOD>_picked_otus/]

- This is the location where the resulting output should be written.

-r REFSEQS_FP, --refseqs_fp=REFSEQS_FP [Default: none]

- This is the path to the reference sequences which blast should compare against, when using the "-m blast" option. This option only works with the blast method.

-b BLAST_DB, --blast_db=BLAST_DB [Default: none]

- This is the path to the pre-existing database which blast should compare against, when using "-m blast" option. This should be the prefix to the database, since multiple files comprise a blast database. This option only works with the blast method.

-s SIMILARITY, --similarity=SIMILARITY [Default: 0.97]

- This is the sequence similarity threshold for picking otus. This option only works with cdhit, blast and mothur

methods.

-e MAX_E_VALUE, --max_e_value=MAX_E_VALUE [Default: 1e-10]

- This is the maximum E-value allowed, when clustering using the "-m blast" option. This option only works with the blast method.

-n PREFIX_PREFILTER_LENGTH, --prefix_prefilter_length=PREFIX_PREFILTER_LENGTH [Default: none]

- This is the length of the sequence prefix, by which to prefilter data. Sequences with identical prefixes using the defined length will be automatically grouped into a single OTU. This is useful for large sequence collections where OTU picking doesn't scale well. A recommended value to use is: 100. This option only works with the cdhit method.

-t, --trie_prefilter [Default: False]

- When using the "-t" option, the longest sequence within a OTU cluster is chosen, however if you use the "-n" option, this option will be deprecated. This option useful for large sequence collections where OTU picking doesn't scale well. This option only works with the cdhit method.

-p PREFIX_LENGTH, --prefix_length=PREFIX_LENGTH [Default: 50]

- This is the prefix length when using the prefix_suffix otu picker. WARNING: This is currently different from prefix_prefilter_length (-n)!

-u SUFFIX_LENGTH, --suffix_length=SUFFIX_LENGTH [Default: 50]

- This is the suffix length when using the prefix_suffix otu picker.

The primary inputs for ***pick_otus.py*** are (i) a FASTA file containing sequences to be clustered; (ii) an OTU threshold (default is 0.97, roughly corresponding to species-level OTUs); (iii) and the method to be applied for clustering sequences into OTUs.

A standard input file for ***pick_otus.py*** should look something like the following:

```
>seq1 some sequence description
ACCGGATATGAGAGAGAGAG
>seq2 another sequence description
ACCGTTATATGAGAGGAG
>seq3
ACCGTTATATTTAATTGGAGAAG
```

Note that sequence identifier lines correspond to the FASTA format, where each line begins with a '>', followed by an optional space character, followed by the sequence identifiers, and then optionally followed by a space character and a comment describing the sequence. Only the first required sequence identifier field (i.e., seq1, seq2, seq3 in the above example) is used by the OTU picker to identify each sequence.

Output:

The output consists of two files (i.e. seqs_otus.txt and seqs_otus.log). The .txt file is composed of tab-delimited lines, where the first field on each line corresponds to an (arbitrary) cluster identifier, and the remaining fields correspond to sequence identifiers assigned to that cluster. Sequence identifiers correspond to those provided in the input FASTA file.

Example lines from the resulting .txt file:

```
0 seq1 seq5
1 seq2
2 seq3
3 seq4 seq6 seq7
```

This result implies that four clusters were created based on 7 input sequences. The first cluster (cluster id 0) contains two

sequences, sequence ids seq1 and seq5; the second cluster (cluster id 1) contains one sequence, sequence id seq2; the third cluster (cluster id 2) contains one sequence, sequence id seq3, and the final cluster (cluster id 3) contains three sequences, sequence ids seq4, seq6, and seq7.

The resulting .log file contains a list of parameters passed to the ***pick_otus.py*** script along with the output location of the resulting .txt file.

Example (cd-hit method):

Using the seqs.fna file generated from ***split_libraries.py*** and outputting the results to the directory "picked_otus/", while using default parameters (cd-hit, 0.97 sequence similarity, no prefix filtering):

```
$ python $qdir/pick_otus.py -i seqs.fna -o picked_otus/
```

Currently the cd-hit OTU picker allows for users to perform a pre-filtering step, so that highly similar sequences are clustered prior to OTU picking. This works by collapsing sequences which begin with an identical n-base prefix, where n is specified by the -n parameter. A commonly used value here is 100 (e.g., -n 100). So, if using this filter with -n 100, all sequences which are identical in their first 100 bases will be clustered together, and only one representative sequence from each cluster will be passed to cd-hit. This is used to greatly increase the run-time of cd-hit-based OTU picking when working with very large sequence collections, as shown by the following command:

```
$ python $qdir/pick_otus.py -i seqs.fna -o picked_otus/ -n 100
```

Alternatively, if the user would like to collapse identical sequences, or those which are subsequences of other sequences prior to OTU picking, they can use the trie prefiltering ("-t") option as shown by the following command:

```
$ python $qdir/pick_otus.py -i seqs.fna -o picked_otus/ -t
```

Note: It is highly recommended to use one of the prefiltering methods when analyzing large dataset (>100,000 seqs) to reduce run-time.

BLAST OTU-Picking Example:

OTUs can be picked against a reference database using the BLAST OTU picker. This is useful, for example, when different regions of the SSU RNA have sequenced and a sequence similarity based approach like cd-hit therefore wouldn't work. When using the BLAST OTU picking method, the user must supply either a reference set of sequences or a reference database to compare against. The OTU identifiers resulting from this step will be the sequence identifiers in the reference database. This allows for use of a pre-existing tree in downstream analyses, which again is useful in cases where different regions of the 16s gene have been sequenced.

The following command can be used to blast against a reference sequence set, using the default E-value and sequence similarity (0.97) parameters:

```
$ python $qdir/pick_otus.py -i seqs.fna -o picked_otus/ -m blast -r ref_seq_set.fna
```

If you already have a pre-built BLAST database, you can pass the database prefix as shown by the following command:

```
$ python $qdir/pick_otus.py -i seqs.fna -o picked_otus/ -m blast -b ref_database
```

If the user would like to change the sequence similarity ("-s") and/or the E-value ("-e") for the blast method, they can use the following command:

```
$ python $qdir/pick_otus.py -i seqs.fna -o picked_otus/ -s 0.90 -e 1e-30
```

Prefix-suffix OTU Picking Example:

OTUs can be picked by collapsing sequences which being and/or end with identical bases (i.e., identical prefixes or suffixes). This OTU picker is currently likely to be of limited use on its own, but will be very useful in collapsing very similar sequences in a chained OTU picking strategy that is currently in development. For example, user will be able to pick OTUs with this method, followed by representative set picking, and then re-pick OTUs on their representative set. This will

allow for highly similar sequences to be collapsed, followed by running a slower OTU picker. This ability to chain OTU pickers is *not yet supported in QIIME*. The following command illustrates how to pick OTUs by collapsing sequences which are identical in their first 50 and last 25 bases:

```
$ python $qdir/pick_otus.py –i seqs.fna –o picked_otus –p 50 –u 25
```

3.3.2 Pick Representative Sequences

QIIME script: ***pick_rep_set.py***

After picking OTUs, you can then pick a representative set of sequences. For each OTU, you will end up with one sequence that can be used in subsequent analyses. By default, the representative sequence for an OTU is chosen as the most abundant sequence showing up in that OTU. This is computed by collapsing identical sequences, and choosing the one that was read the most times as the representative sequence (note that each of these would have a different sequence identifier in the FASTA provided as input).

Usage: ***pick_rep_set.py*** [options]

Input Arguments:

-i OTU_FP, --input_file=OTU_FP [REQUIRED]

- Path the OTU file containing OTUs and the sequence identifiers which correspond to a particular OTU (i.e., resulting OTU file from ***pick_otus.py***)

-f FASTA_FP, --fasta_file=FASTA_FP [REQUIRED]

- Path to FASTA file containing sequences, where the sequence ids may contain the sample ids (i.e., resulting FASTA file from ***split_libraries.py***)

-m REP_SET_PICKING_METHOD, --rep_set_picking_method=REP_SET_PICKING_METHOD [default: most_abundant]

- This is the method for picking representative sequences for each OTU. Alternative methods for choosing representative sequences are to choose the longest sequence in an OTU ("-m longest"); the first sequence listed in an OTU ("-m first"); or a random sequence from the OTU ("-m random").

-o RESULT_FP, --result_fp=RESULT_FP [Default: <input_sequences_filepath>_rep_set.fasta]

- This is the location where the resulting output should be written.

-l LOG_FP, --log_fp=LOG_FP [Default: No log file created.]

- This is the location where the resulting .log file should be written.

-s SORT_BY, --sort_by=SORT_BY [Default: otu]

- This is the method by which the resulting representative sequence set is sorted. The options to sort by are by OTU ("-s otu") or by the sequence identifier ("-s seq_id")

Output:

The output from ***pick_rep_set.py*** is a single FASTA file containing one sequence per OTU. The FASTA header lines will be the OTU identifier (from here on used as the unique sequence identifier) followed by a space, followed by the sequence identifier originally associated with the representative sequence. The name of the output FASTA file will be <input_sequences_filepath>_rep_set.fasta by default, or can be specified via the "-o" parameter.

Examples:

The script ***pick_rep_set.py*** takes as input an 'OTU file' (via the "-i" parameter) which maps OTU identifiers to sequence

identifiers. Typically, this will be the output file provided by ***pick_otus.py***. Additionally, a FASTA file is required, via "-f", which contains all of the sequences whose identifiers are listed in the OTU file. The following command shows an example of this where the resulting file is output to the directory "repr_set/" and default parameters were used (choose most abundant, sort by OTU id and do not write a log file):

```
$ python $qdir/pick_rep_set.py -i seqs_otus.txt -f seqs.fna -o repr_set/
```

Alternatively, if the user would like to choose the sequence by random "-m random" and then sort by the sequence identifier ("-s seq_id"), they could use the following command:

```
$ python $qdir/pick_rep_set.py -i seqs_otus.txt -f seqs.fna -o repr_set/ -m random -s seq_id
```

3.4. Chimera Checking

QIIME script: ***identify_chimeric_seqs.py***

A FASTA file of sequences, can be screened to remove chimeras (sequences generated due to the PCR amplification of multiple templates or parent sequences). QIIME currently includes a single taxonomy-assignment-based approach, blast_fragments, for identifying sequences as chimeric.

Usage: ***identify_chimeric_seqs.py*** [options]

Input Arguments:

-i INPUT_SEQS_FP, --input_seqs_fp=INPUT_SEQS_FP [REQUIRED]

- Path to FASTA file containing sequences to be assigned (i.e., resulting FASTA file from ***pick_rep_set.py***)

-t ID_TO_TAXONOMY_FP, --id_to_taxonomy_fp=ID_TO_TAXONOMY_FP [Default: none; REQUIRED when method is blast_fragments]

- Path to tab-delimited file which maps sequences to an assigned taxonomy. Each assigned taxonomy is provided as a comma-separated list.

-r REFERENCE_SEQS_FP, --reference_seqs_fp=REFERENCE_SEQS_FP [Default: none; must provide either –reference_seqs_fp or –blast_db when method is blast_fragments]

- This is the path to the reference sequences (used to build a blast database).

-b BLAST_DB, --blast_db=BLAST_DB [Default: none; must provide either –reference_seqs_fp or –blast_db when method is blast_fragments]

- Database to blast against.

-m CHIMERA_DETECTION_METHOD, --chimera_detection_method=CHIMERA_DETECTION_METHOD [Default:blast_fragments]

- This is the chimera detection method that should be used.

-n NUM_FRAGMENTS, --num_fragments=NUM_FRAGMENTS [Default: 3]

- This is the number of fragments that sequences should be split in to (i.e., number of expected breakpoints + 1).

-d TAXONOMY_DEPTH, --taxonomy_depth=TAXONOMY_DEPTH [Default: 4]

- This is the Number of taxonomic divisions to consider when comparing taxonomy assignments.

-e MAX_E_VALUE, --max_e_value=MAX_E_VALUE [default: 1e-30]

- This is the maximum E-value allowed, when clustering using the "-m blast_fragments" option. This option only works with the blast method.

-o OUTPUT_FP, --output_fp=OUTPUT_FP [Default: derived from input_seqs_fp]

- This is the location where the resulting output should be written.

-v, --verbose [default: False]

- If this parameter is passed, then the script will print information during execution, which is useful for debugging.

Output:

The result of ***identify_chimeric_seqs.py*** is text file which identifies which sequences are chimeric.

Example:

The blast_fragments chimera detection method is the default method used ***by identify_chimeric_sequences.py***. For each sequence provided as input, the blast_fragments method splits the input sequence into n roughly-equal-sized, non-overlapping fragments, and assigns taxonomy to each fragment against a reference database. The BlastTaxonAssigner (implemented in qiime.assign_taxonomy) is used for this. The taxonomies of the fragments are compared with one another (at a default depth of 4), and if contradictory assignments are returned the sequence is identified as chimeric. For example, if an input sequence was split into 3 fragments, and the following taxon assignments were returned:

```
fragment1: Archaea;Euryarchaeota;Methanobacteriales;Methanobacterium
fragment2: Archaea;Euryarchaeota;Halobacteriales;uncultured
fragment3: Archaea;Euryarchaeota;Methanobacteriales;Methanobacterium
```

The sequence would be considered chimeric at a depth of 3 (Methanobacteriales vs. Halobacteriales), but non-chimeric at a depth of 2 (all Euryarchaeota).

blast_fragments begins with the assumption that a sequence is non-chimeric, and looks for evidence to the contrary. This is important when, for example, no taxonomy assignment can be made because no blast result is returned. If a sequence is split into three fragments, and only one returns a blast hit, that sequence would be considered non-chimeric. This is because there is no evidence (i.e., contradictory blast assignments) for the sequence being chimeric. This script can be run by the following command, where the resulting data is written to the directory "identify_chimeras/" and using default parameters (e.g. chimera detection method ("-m blast_fragments"), number of fragments ("-n 3"), taxonomy depth ("-d 4") and maximum E-value ("-e 1e-30")):

```
$ python $qdir/identify_chimeric_sequences.py -i repr_set_seqs.fasta -t taxonomy_assignment.txt -r
        ref_seq_set.fna -o identify_chimeras/
```

3.5. Eliminating Human (or Other) Contamination

3.5.1 Exclude Sequences by BLAST

QIIME script: ***exclude_seqs_by_blast.py***

This code is designed to allow users of the QIIME workflow to conveniently exclude unwanted sequences from their data. This is mostly useful for excluding human sequences from runs to comply with Internal Review Board (IRB) requirements, but may also have other uses (e.g. perhaps excluding a major bacterial contaminant). Sequences from a run are searched against a user-specified subject database, where BLAST hits are screened by e-value and the percentage of the query that aligns to the sequence.

Usage: ***exclude_seqs_by_blast.py*** [options]

Input Arguments:

-i QUERYDB, --querydb=QUERYDB [REQUIRED]

- Path to FASTA file containing sequences that are used as the query (i.e., resulting FASTA file from ***pick_rep_set.py***).

-d SUBJECTDB, --subjectdb=SUBJECTDB [REQUIRED]

- This is the path to the reference sequences which blast should compare against.

-o OUTPUTFILENAME, --outputfilename=OUTPUTFILENAME [REQUIRED]

- This is the prefix that should be used on all resulting files, such that sequences passing the screen, failing the screen, raw BLAST results and the log will be saved to your filename_prefix + '.screened', '.excluded', '.raw_blast_results', and '.sequence_exclusion_log', respectively.

-e E_VALUE, --e_value=E_VALUE [Default: 1e-10]

- This is the maximum E-value allowed, when clustering using the BLAST method.

-p PERCENT_ALIGNED, --percent_aligned=PERCENT_ALIGNED [Default: 0.97]

- The percent (%) aligned cutoff for the query sequence coverage when aligned to sequences in the BLAST database.

--blastmatroot=BLASTMATROOT [Default: none]

- This is the path to the folder containing the BLAST matrices (blastmat).

--working_dir=WORKING_DIR [Default: /tmp]

- This is the path to the working directory for BLAST.

-M MAX_HITS, --max_hits=MAX_HITS [Default: 100]

- This is the maximum number of BLAST hits. CAUTION: Filtering on percent aligned ("-p") occurs after BLAST and if the user sets the max value to 1 valid contaminants could be missed.

-W WORDSIZE, --word_size=WORDSIZE [Default: 28]

- This is the word size to use for the BLAST search.

--debug [Default: False]

- If this parameter is True, then the script will print information during execution, which is useful for debugging.

Output:

Four output files are generated based on the supplied outputpath + unique suffixes:

1. "filename_prefix".screened: A FASTA file of sequences that did pass the screen (i.e. matched the database and passed all filters).
2. "filename_prefix".excluded: A FASTA file of sequences that did not pass the screen.
3. "filename_prefix".raw_blast_results: Contains the raw BLAST results from the screening.
4. "filename_prefix".sequence_exclusion_log: A log file summarizing the options used and results obtained.

Examples:

The following is a simple example, where the user can take a given FASTA file (i.e. resulting FASTA file from ***pick_rep_set.py***) and blast those sequences against a reference FASTA file containing the set of sequences which are considered contaminated:

```
$ python $qdir/exclude_seqs_by_blast.py -i repr_set_seqs.fasta -d ref_seq_set.fna -o exclude_seqs/
```

Alternatively, if the user would like to change the percent of aligned sequence coverage ("-p") or the maximum E-value ("-e"), they can use the following command:

```
$ python $qdir/exclude_seqs_by_blast.py -i repr_set_seqs.fasta -d ref_seq_set.fna -o exclude_seqs/ -p 0.95 -e
    1e-10
```

3.5.2 Filter OTUs by Removing Samples

QIIME script: ***filter_otus_by_sample.py***

This filter allows for the removal of sequences and OTUs containing user-specified Sample IDs, for instance, the removal of negative control samples. This script identifies OTUs containing the specified Sample IDs and removes its corresponding sequence from the sequence collection.

Usage: ***filter_otus_by_sample.py*** [options]

Input Arguments:

-i INPUT_OTU_PATH, --input_otu_path=INPUT_OTU_PATH [REQUIRED]

- Path to OTU file containing sequence ids assigned to each OTU (i.e., resulting OTU file from ***pick_otus.py***).

-f FASTA_FILE, --fasta_file=FASTA_FILE [REQUIRED]

- Path to FASTA file containing all sequences (i.e., resulting FASTA file from ***split_libraries.py***).

-s SAMPLES_TO_EXTRACT, --samples_to_extract=SAMPLES_TO_EXTRACT [REQUIRED]

- This is a list of sample ids, which should be removed from the OTU file.

-o DIR_PATH, --dir_path=DIR_PATH [Default=]

- This is the location where the resulting output should be written.

Output:

As a result a new OTU and sequence file is generated and written to a randomly generated folder where the name of the folder starts with "filter_by_otus...." Also included in the folder, is another FASTA file containing the removed sequences, leaving the user with 3 files.

Example:

The following command can be used, where all options are passed (using the resulting OTU file from ***pick_otus.py***, FASTA file from ***split_libraries.py*** and removal of Sample1) with the resulting data being written to the output directory "filtered_otus/":

```
$ python $qdir/filter_otus_by_sample.py -i seqs_otus.txt -f seqs.fna -s Sample1 -o filtered_otus/
```

3.5.3 Filter OTUs using Metadata

QIIME script: ***filter_ by_metadata.py***

This filter allows for the removal of sequences and OTUs that either do or don't match specified metadata, for instance, isolating samples from a specific set of studies or body sites. This script identifies samples matching the specified metadata criteria, and outputs a filtered mapping file and OTU table containing only the specified samples.

Usage: ***filter_by_metadata.py*** [options]

Input Arguments:

-i OTU_FNAME, --otu=OTU_FNAME

- Path to OTU file containing sequence ids assigned to each OTU (i.e., resulting OTU file from ***pick_otus.py***).

-m MAP_FNAME, --map=MAP_FNAME

- This is the user-generated mapping file.

-o OTU_OUT_FNAME, --otu_outfile=OTU_OUT_FNAME [Default: otu_filename.filtered.xls]

- This is the filename, where the resulting otu file should be written.

-p MAP_OUT_FNAME, --map_outfile=MAP_OUT_FNAME [Default: map_filename.filtered.xls]

- This is the filename, where the resulting mapping file should be written.

-s VALID_STATES, --states=VALID_STATES

- This is a string containing valid states, e.g. 'STUDY_NAME:DOG'

-n NUM_SEQS_PER_OTU, --num_seqs_per_otu=NUM_SEQS_PER_OTU

- This is the minimum counts across samples in order to keep an OTU, e.g. 5.

Output:

The result is a filtered OTU table and mapping file meeting the desired criteria.

Example:

The following command can be used, where all options are passed (using the resulting OTU file from ***pick_otus.py***, the original Fasting_Map file, and keeping only the Control sequences in the Treatment field) with the resulting data being written to seqs_otus.txt.filtered.xls and Fasting_Map.txt.filtered.xls:

```
$ python $qdir/filter_otus_by_sample.py -i seqs_otus.txt -m Fasting_Map.txt -s 'Treatment:Control'
```

Some variations (not so useful on this dataset, but more useful on larger datasets) are:

- Keeping both Control and Fast in the Treatment field (i.e. keeping everything):

```
$ python $qdir/filter_otus_by_sample.py -i seqs_otus.txt -m Fasting_Map.txt -s 'Treatment:Control,Fast'
```

- Excluding Fast in the Treatment field (same as the first example) – the syntax here is * to keep everything, then !Fast to eliminate the Fast group:

```
$ python $qdir/filter_otus_by_sample.py -i seqs_otus.txt -m Fasting_Map.txt -s 'Treatment:*,!Fast'
```

- Keeping only samples with both Control in the Treatment field and 20061218 in the DOB field:

$ python $qdir/filter_otus_by_sample.py -i seqs_otus.txt -m Fasting_Map.txt -s 'Treatment:Control,DOB:20061218'

- Keeping only samples with Control in the Treatment field and OTUs with counts of at least 5 across samples:

$ python $qdir/filter_otus_by_sample.py -i seqs_otus.txt -m Fasting_Map.txt –s 'Treatment:Control' –n 5

Note that the filtered mapping file will automatically exclude any columns that are the same for all the samples that are left, and will also exclude (except for SampleID) any columns that are different for all the samples that are left, making it more useful for downstream analyses with the coloring tools.

3.6. Align Sequences and Filter Alignment

3.6.1 Sequence Alignment

QIIME script: ***align_seqs.py***

This script aligns the sequences in a FASTA file to each other or to a template sequence alignment, depending on the method chosen. Currently, there are three methods which can be used by the user:

1. PyNAST (Caporaso et al., 2009) - The default alignment method is PyNAST, a python implementation of the NAST alignment algorithm. The NAST algorithm aligns each provided sequence (the "candidate" sequence) to the best-matching sequence in a pre-aligned database of sequences (the "template" sequence). Candidate sequences are not permitted to introduce new gap characters into the template database, so the algorithm introduces local mis-alignments to preserve the existing template sequence.
2. MUSCLE (Edgar, 2004) - MUSCLE is an alignment method which stands for MUltiple Sequence Comparison by Log-Expectation.
3. INFERNAL (Nawrocki, Kolbe, & Eddy, 2009) - Infernal ("INFERence of RNA ALignment") is for an alignment method for using RNA structure and sequence similarities.

Usage: ***align_seqs.py*** [options]

Input Arguments:

-i INPUT_FASTA_FP, --input_fasta_fp=INPUT_FASTA_FP [REQUIRED]

- Path to FASTA file containing sequences to be aligned (i.e., resulting FASTA file from ***pick_rep_set.py***).

-t TEMPLATE_FP, --template_fp=TEMPLATE_FP [Default: none; REQUIRED if -m pynast or -m infernal]

- This is the path to the reference sequences which should be compare against, when using the "-m pynast" or "-m infernal" option. If you'd like to align against the greengenes core set, you can download the template alignment from here: http://greengenes.lbl.gov/Download/Sequence_Data/Fasta_data_files/core_set_aligned.fasta.imputed

-m ALIGNMENT_METHOD, --alignment_method=ALIGNMENT_METHOD [Default: pynast]

- This is the method for aligning the sequences.

-a PAIRWISE_ALIGNMENT_METHOD, --pairwise_alignment_method=PAIRWISE_ALIGNMENT_METHOD [Default: blast]

- This is the method for performing pairwise alignment in PyNAST. Several methods are available to align the candidate sequence to the template sequence. They have minor differences in speed and quality of results (by default, the program bl2seq is used).

-d BLAST_DB, --blast_db=BLAST_DB [Default: created on-the-fly from template_alignment]

- This is the path to the pre-existing database which blast should compare against, when using "-m pynast"

option. This should be the prefix to the database, since multiple files comprise a blast database.

-b BLAST_EXECUTABLE, --blast_executable=BLAST_EXECUTABLE [Default: /software/bin/blastall]

- This is the path to blast executable when using the "-m pynast" option.

-o OUTPUT_DIR, --output_dir=OUTPUT_DIR [Default: <ALIGNMENT_METHOD>_aligned]

- This is the location where the resulting output should be written.

-e MIN_LENGTH, --min_length=MIN_LENGTH [Default: 150]

- This is the minimum sequence length for a sequence to be included in the alignment.

-p MIN_PERCENT_ID, --min_percent_id=MIN_PERCENT_ID [Default: 75.0]

- This is the minimum percent sequence identity closest to a particular blast hit for a sequence to be included in the alignment.

Output:

All aligners will output a fasta file containing the alignment and log file in the directory specified by –output_dir (default <alignment_method>_aligned). PyNAST additionally outputs a failures file, containing the sequences which failed to align. So the result of ***align_seqs.py*** will be up to three files, where the prefix of each file depends on the user supplied FASTA file:

1. "..._aligned.fasta" - This is a FASTA file containing all aligned sequences.
2. "..._failures.fasta - This is a FASTA file containing all sequences which did not meet all the criteria specified. (PyNAST only)
3. "..._log.txt" - This is a log file containing information pertaining to the results obtained from a particular method (e.g. BLAST percent identity, etc.).

Alignment with PyNAST

The default alignment method is PyNAST, a python implementation of the NAST alignment algorithm. The NAST algorithm aligns each provided sequence (the "candidate" sequence) to the best-matching sequence in a pre-aligned database of sequences (the "template" sequence). Candidate sequences are not permitted to introduce new gap characters into the template database, so the algorithm introduces local mis-alignments to preserve the existing template sequence. The quality thresholds are the minimum requirements for matching between a candidate sequence and a template sequence. The set of matching template sequences will be searched for a match that meets these requirements, with preference given to the sequence length. By default, the minimum sequence length is 150 and the minimum percent id is 75%. The minimum sequence length is much too long for typical pyrosequencing reads, but was chosen for compatibility with the original NAST tool.

The following command can be used for aligning sequences using the PyNAST method, where we supply the program with a FASTA file of unaligned sequences (i.e. resulting FASTA file from ***pick_rep_set.py***, a FASTA file of pre-aligned sequences (this is the template file, which is typically the Greengenes core set - available from http://greengenes.lbl.gov/), and the results will be written to the directory "pynast_aligned/":

```
$ python $qdir/align_seqs.py -i repr_set_seqs.fasta -t core_set_template.fasta -o pynast_aligned/
```

Alternatively, one could change the minimum sequence length ("-e") requirement and minimum sequence identity ("-p"), using the following command:

```
$ python $qdir/align_seqs.py -i repr_set_seqs.fasta -t core_set_template.fasta -o pynast_aligned/ -e 500 -p 95.0
```

Alignment with MUSCLE

One could also use the MUSCLE algorithm. The following command can be used to align sequences (i.e. the resulting FASTA file from ***pick_rep_set.py***), where the output is written to the directory "muscle_alignment/":

```
$ python $qdir/align_seqs.py -i repr_set_seqs.fasta -o muscle_alignment/
```

Alignment with Infernal

An alternative alignment method is to use Infernal. Infernal is similar to the PyNAST method, in that you supply a template alignment, although Infernal has several distinct differences. Infernal takes a multiple sequence alignment with a corresponding secondary structure annotation. This input file must be in Stockholm alignment format. There is a fairly good description of the Stockholm format rules at: http://en.wikipedia.org/wiki/Stockholm_format. Infernal will use the sequence and secondary structural information to align the candidate sequences to the full reference alignment. Similar to PyNAST, Infernal will not allow for gaps to be inserted into the reference alignment. Using Infernal is slower than other methods, and therefore is best used with sequences that do not align well using PyNAST.

The following command can be used for aligning sequences using the Infernal method, where we supply the program with a FASTA file of unaligned sequences, a STOCKHOLM file of pre-aligned sequences and secondary structure (this is the template file – an example file can be obtained from: http://tajmahal.colorado.edu/tmp/tmprEJYWsLiGSseed.16s.reference_model.sto.zip), and the results will be written to the directory "infernal_aligned/":

```
$ python $qdir/align_seqs.py -m infernal -i repr_set_seqs.fasta -t seed.16s.reference_model.sto -o
        infernal_aligned/
```

3.6.2 Filtering Sequence Alignment

QIIME script: ***filter_alignment.py***

The filter_alignment.py script should be applied to generate a useful tree when aligning against a template alignment (e.g., with PyNAST). This script will remove positions which are gaps in every sequence (common for PyNAST, as typical sequences cover only 200-400 bases, and they are being aligned against the full 16S gene). Additionally, the user can supply a lanemask file, that defines which positions should included when building the tree, and which should be ignored. Typically, this will differentiate between non-conserved positions, which are uninformative for tree building, and conserved positions which are informative for tree building. FILTERING ALIGNMENTS WHICH WERE BUILD WITH PYNAST AGAINST THE GREENGENES CORE SET ALIGNMENT SHOULD BE CONSIDERED AN ESSENTIAL STEP.

Usage***: filter_alignment.py*** [options]

Input Arguments:

-i INPUT_FASTA_FILE, --input_fasta_file=INPUT_FASTA_FILE [REQUIRED]

- Path to FASTA file containing the aligned sequences (i.e., resulting FASTA file from ***align_seqs.py***).

-o OUTPUT_DIR, --output_dir=OUTPUT_DIR [Default: .]

- This is the location where the resulting output should be written.

-m LANE_MASK_FP, --lane_mask_fp=LANE_MASK_FP [Default: none]

- This is the path to lanemask file – if a PyNAST alignment was performed against the greengenes core set, this can be obtained from the Greengenes website (available from http://greengenes.lbl.gov/Download/Sequence_Data/lanemask_in_1s_and_0s)

-g ALLOWED_GAP_FRAC, --allowed_gap_frac=ALLOWED_GAP_FRAC [Default: 0.999999]

- This is the gap filter threshold. Filtering is performed when the number of positions in a column are gaps is greater than the allowed gap fraction of the sequences

--verbose [Default: none]

- If this parameter is used, then the script will print information during execution, which is useful for debugging.

Output:

The output of ***filter_alignment.py*** consists of a single FASTA file, which ends with "_pfiltered.fasta", where the "p" stands for positional filtering of the columns.

Example:

As a simple example of this script, the user can use the following command, which consists of an input FASTA file (i.e. resulting file from ***align_seqs.py***), lanemask template file and the output directory "filtered_alignment/":

```
$ python $qdir/filter_otus_by_sample.py -i repr_set_seqs_aligned.fna -m lanemask_template -o
    filtered_alignment/
```

Alternatively, if the user would like to use a different gap fraction threshold ("-g"), they can use the following command:

```
$ python $qdir/filter_otus_by_sample.py -i repr_set_seqs_aligned.fna -m lanemask_template -o
    filtered_alignment/ -g 0.95
```

3.7. Phylogeny

QIIME script***: make_phylogeny.py***

Many downstream analyses require that the phylogenetic tree relating the OTUs in a study be present. The script ***make_phylogeny.py*** produces this tree from a multiple sequence alignment. Trees are constructed with a set of sequences representative of the OTUs, using FastTree (Price, Dehal, & Arkin, 2009). For grafted trees, subtrees are built with all sequences in an OTU and then grafted onto the representative set tree.

Usage: ***make_phylogeny.py*** [options]

Input Arguments:

-t TREE_METHOD, --tree_method=TREE_METHOD [Default: fasttree]

- This is the method that should be used for building the phylogenetic tree. Valid choices are: clearcut, clustalw, raxml, fasttree_v1, fasttree, muscle and mafft.

-o RESULT_FP, --result_fp=RESULT_FP [Default: <input_sequences_filename>.tre]

- This is the location where the resulting output should be written.

-l LOG_FP, --log_fp=LOG_FP [Default: No log file created.]

- This is the location where the resulting .log file should be written.

-i INPUT_FP, --input_fp=INPUT_FP [REQUIRED]

- Path to FASTA file containing the aligned sequences (i.e., resulting FASTA file from ***filter_alignment.py***).

Output:

The result of ***make_phylogeny.py*** consists of a tree file (.tre) and a log file. The tree file is formatted using the Newick format and this file can be viewed using most tree visualization tools, such as TopiaryTool, FigTree, etc..

Examples:

A simple example of ***make_phylogeny.py*** is shown by the following command, where we use the default tree building method (fasttree) and write the file to the current working directory without a log file:

```
$ python $qdir/make_phylogeny.py -i repr_set_seqs_aligned_pfiltered.fasta
```

Alternatively, if the user would prefer using another tree building method (i.e. clearcut (Sheneman, Evans, & Foster, 2006)), then they could use the following command:

```
$ python $qdir/make_phylogeny.py -i repr_set_seqs_aligned_pfiltered.fasta -t clearcut
```

Note: For whichever method used, the 3rd party program must be properly installed on the user's computer.

3.8. Taxonomy Assignment

QIIME script: ***assign_taxonomy.py***

Given a set of sequences, assign_taxonomy attempts to assign the taxonomy of each sequence. Currently there are two methods implemented: assignment with BLAST and assignment with the RDP classifier. The output of this step is a mapping of sequence identifiers to taxonomy with a quality score.

Usage: ***assign_taxonomy.py*** [options]

Input Arguments:

-i INPUT_SEQS_FP, --input_seqs_fp=INPUT_SEQS_FP [REQUIRED]

- Path to FASTA file containing sequences to be aligned (i.e., resulting FASTA file from ***pick_rep_set.py***).

-t ID_TO_TAXONOMY_FP, --id_to_taxonomy_fp=ID_TO_TAXONOMY_FP [Default: none; REQUIRED when method is blast]

- This is the path to tab-delimited file which maps sequences to their corresponding taxonomy, where each assigned taxonomy is provided as a comma-separated list. For taxonomy assignment with rdp, each assigned taxonomy must be exactly 6 levels deep.

-r REFERENCE_SEQS_FP, --reference_seqs_fp=REFERENCE_SEQS_FP [Default: none; The user must provide either --blast_db or --reference_seqs_db for assignment with blast.]

- This is the path to the reference sequences which blast should compare against. For assignment with blast, these are used to generate a blast database. For assignment with rdp, they are used as training sequences for the classifier.

-m ASSIGNMENT_METHOD, --assignment_method=ASSIGNMENT_METHOD [Default: blast]

- This is the taxonomy assignment method that should be used.

-b BLAST_DB, --blast_db=BLAST_DB [Default: none; The user must provide either --blast_db or --reference_seqs_db for assignment with blast.]

- This is the path to the pre-existing database which blast should compare against, when using "-m blast" option. This should be the prefix to the database, since multiple files comprise a blast database.

-c CONFIDENCE, --confidence=CONFIDENCE [Default: 0.8]

- This is the minimum confidence needed to record an assignment and can only be used for the rdp method. Assignments must exceed this score to be written in the output file.

-e E_VALUE, --e_value=E_VALUE [Default: 0.001]

- This is the maximum E-value allowed, when assigning using the BLAST method.

-o OUTPUT_DIR, --output_dir=OUTPUT_DIR [Default: <ASSIGNMENT_METHOD>_assigned_taxonomy]

- This is the location where the resulting output should be written.

Output:

The consensus taxonomy assignment implemented here is the most detailed lineage description shared by 90% or more of the sequences within the OTU (this level of agreement can be adjusted by the user). The full lineage information for each sequence is one of the output files of the analysis. In addition, a conflict file records cases in which a phylum-level taxonomy assignment disagreement exists within an OTU (such instances are rare and can reflect sequence misclassification within the greengenes database).

Example of consensus lineage:

The OTU containing 5 sequences annotated as shown below would be assigned to the "Desulfovibrionaceae" level because only 80% of sequences agree with the "LE30" annotation.

```
Bacteria; Proteobacteria; Desulfovibrionales; Desulfovibrionaceae; LE30
Bacteria; Proteobacteria; Desulfovibrionales; Desulfovibrionaceae; LE30
Bacteria; Proteobacteria; Desulfovibrionales; Desulfovibrionaceae; LE30
Bacteria; Proteobacteria; Desulfovibrionales; Desulfovibrionaceae;
Bacteria; Proteobacteria; Desulfovibrionales; Desulfovibrionaceae; LE30
```

Assignments are provided in a two column tab-delimited format, which maps input sequence identifiers to assignments. Each assignment is specified as a list of taxa separated by a ';' character.

Example of an assignment output file:

```
AY800210 Archaea;Euryarchaeota;Halobacteriales;uncultured
EU883771 Archaea;Euryarchaeota;Methanomicrobiales;Methanomicrobium et rel.
EF503699 Archaea;Crenarchaeota;uncultured;uncultured
DQ260310 Archaea;Euryarchaeota;Methanobacteriales;Methanobacterium
EF503697 Archaea;Crenarchaeota;uncultured;uncultured
```

Sample Assignment with BLAST

Taxonomy assignments are made by searching input sequences against a blast database of pre-assigned reference sequences. If a satisfactory match is found, the reference assignment is given to the input sequence. This method does not take the hierarchical structure of the taxonomy into account, but it is very fast and flexible. If a file of reference sequences is provided, a temporary blast database is built on-the-fly. The quality scores assigned by the BLAST taxonomy assigner are e-values.

To assign the sequences to the representative sequence set, using a reference set of sequences and a taxonomy to id assignment text file, where the results are output to default directory "blast_assigned_taxonomy", you can run the following command:

```
$ python $qdir/assign_taxonomy.py -i repr_set_seqs.fasta -r ref_seq_set.fna -t id_to_taxonomy.txt
```

Optionally, the user could changed the E-value ("-e"), using the following command:

```
$ python $qdir/assign_taxonomy.py -i repr_set_seqs.fasta -r ref_seq_set.fna -t id_to_taxonomy.txt -e 0.01
```

Assignment with the RDP Classifier

The RDP Classifier program (Wang, Garrity, Tiedje, & Cole, 2007) assigns taxonomies by matching sequence segments of length 8 to a database of previously assigned sequences. It uses a naive bayesian algorithm, which means that for each potential assignment, it attempts to calculate the probability of the observed matches, assuming that the assignment is correct and that the sequence segments are completely independent. The RDP Classifier is distributed with a pre-built database of assigned sequence, which is used by default. The quality scores provided by the RDP classifier are confidence values.

To assign the representative sequence set, where the output directory is "rdp_assigned_taxonomy", the you can run the following command:

```
$ python $qdir/assign_taxonomy.py -i repr_set_seqs.fasta -m rdp
```

Alternatively, the user could change the minimum confidence score ("-c"), using the following command:

```
$ python $qdir/assign_taxonomy.py -i repr_set_seqs.fasta -m rdp -c 0.85
```

Note: If a reference set of sequences and taxonomy to id assignment file are provided, the script will use them to generate a new training dataset for the RDP Classifier on-the-fly. Due to limitations in the generation of a training set, each provided assignment must contain exactly 6 taxa in the following order: domain (level=2), phylum (level=3), class (level=4), order (5), family (level=6), and genus (level=7). Additionally, each genus name must be unique, due to the internal algorithm used by the RDP Classifier.

3.9 Make OTU Table

3.9.1 Make OTU Table

QIIME script: ***make_otu_table.py***

The script ***make_otu_table.py*** tabulates the number of times an OTU is found in each sample, and adds the taxonomic predictions for each OTU in the last column if a taxonomy file is supplied.

Usage: ***make_otu_table.py*** [options]

Input Arguments:

-i OTU_FNAME, --input_otu_fname=OTU_FNAME [REQUIRED]

- Path to OTU file containing sequence ids assigned to each OTU (i.e., resulting OTU file from ***pick_otus.py***).

-t TAXONOMY_FNAME, --taxonomy=TAXONOMY_FNAME [REQUIRED]

- Path to taxonomy assignment, containing the assignments of taxons to sequences (i.e., resulting txt file from ***assign_taxonomy.py***).

-o OUTPUT_FNAME, --output_fname=OUTPUT_FNAME [Default: stdout]

- This is the filename that should be used when writing the output.

Output:

The output of ***make_otu_table.py*** is a tab-delimited text file, where the columns correspond to Samples and rows correspond to OTUs and the number of times a sample appears in a particular OTU.

Example:

For this example the input is an OTU file containing sequence ids assigned to each OTU (i.e., resulting OTU file from ***pick_otus.py***) and a text file containing the taxonomy assignments (i.e., resulting text file from ***assign_taxonomy.py***), where the output file is defined as "otu_table.txt".

```
$ python $qdir/make_otu_table.py -i seqs_otus.txt -t repr_set_tax_assignments.txt -o otu_table.txt
```

3.9.2 Filter OTU Table

QIIME script: ***filter_otu_table.py***

After the OTU has been generated, the user may want to filter the table based on the number of samples within each OTU or by the number of sequences per OTU. This step is generally done to reduce the noise within the OTU table and can also reduce the overall size of the table, which is essential when performing analyses on large datasets. Along with filtering based on samples or sequences, the user can include and exclude specific taxon groups.

Usage: ***filter_otu_table.py*** [options]

Input Arguments:

-i OTU_FNAME, --otu_filename=OTU_FNAME [REQUIRED]

- This is the path to the OTU table (i.e., the resulting OTU table from ***make_otu_table.py***).

-c MIN_COUNT, --min_count=MIN_COUNT [Default: 1]

- This is the minimum number of sequences that an OTU is present in, for an OTU to be kept in the OTU table.

-s MIN_SAMPLES, --min_samples=MIN_SAMPLES [Default: 2]

- This is the minimum number of Samples that an OTU is present in, for an OTU to be kept in the OTU table.

-t INCLUDE_TAXONOMY, --include_taxonomy=INCLUDE_TAXONOMY [Default=]

- This is a list of taxonomy terms to include in the resulting OTU table.

-e EXCLUDE_TAXONOMY, --exclude_taxonomy=EXCLUDE_TAXONOMY [Default=]

- This is list of taxonomy terms to exclude in the resulting OTU table.

-o DIR_PATH, --dir_path=DIR_PATH [Default=./]

- This is the location where the resulting output should be written.

Output:

The result of ***filter_otu_table.py*** creates a new OTU table, where the filename uses the input OTU filename and appends "_filtered.txt" to the end of the name.

Examples:

To filter the OTU table using the default parameters ("-c 1" and "-s 2"), then write the results to the current working directory, you can use the code as follows:

```
$ python $qdir/filter_otu_table.py -i otu_table.txt
```

To filter by the number of samples (i.e., 5) within each OTU (keep only OTU's found in at least X number of samples), you can use the code as follows:

```
$ python $qdir/filter_otu_table.py -i otu_table.txt -s 5
```

To filter by the number of sequences (i.e., 5) within each OTU (keep only OTU's with at least X sequences in the OTU), you can use the code as follows:

```
$ python $qdir/filter_otu_table.py -i otu_table.txt -c 5
```

To include ("Bacteria") and exclude ("Proteobacteria") certain taxon groups (options -t / -e respectively), you can use the code as follows:

$ python $qdir/filter_otu_table.py -i otu_table.txt -t Bacteria -e Proteobacteria

3.9.3 OTU Significance and Co-occurrence Analysis

QIIME script: ***otu_category_significance.py***

The script ***otu_category_significance.py*** tests whether any of the OTUs in an OTU table are significantly associated with a category in the category mapping file. This code uses either ANOVA or the G test of independence to find OTUs whose members are differentially represented across experimental treatments. It can also be used with presence/absence data for a phylogenetic group (such as that determined with quantitative PCR) to determine if any OTUs co-occur with a taxon of interest.

Usage: ***otu_category_significance.py*** [options]

Input Arguments:

-i OTU_TABLE_FP, --otu_table_fp=OTU_TABLE_FP [REQUIRED]

- This is the path to the OTU table (i.e., the resulting OTU table from ***make_otu_table.py*** or ***filter_otu_table.py***).

-m CATEGORY_MAPPING_FP, --category_mapping_fp=CATEGORY_MAPPING_FP [REQUIRED]

- This is the user-generated mapping file.

-c CATEGORY, --category=CATEGORY [REQUIRED]

- This is the category to test from the user-generated mapping file. The category must match the name of a column header in the mapping file exactly.

-s TEST, --test=TEST

- This is the type of statistical test to run. The options are: g_test and ANOVA. The g_test metric runs the G test of independence. It only considers presence/absence and test the hypothesis that the presence/absence pattern across categories is non-random. The ANOVA metric runs a one-way ANOVA test across the categories. It takes into account differences in OTU relative abundances across categories, testing the null hypothesis that the mean relative abundance is the same across categories.

-o OUTPUT_FP, --output_fp=OUTPUT_FP [Default: otu_category_G_test_results.txt]

- This is the filename to which the output will be written.

-f FILTER, --filter=FILTER [Default: 10]

- This is the minimum number of samples that must contain the OTU for the OTU to be included in the analysis. When this used, the filter is first applied so that only OTUs that were detected in at least a defined number of samples are included in the analysis. This filter is important, since it does not make sense to look for associations when an OTU is only found in 1 or 2 samples.

-t THRESHOLD, --threshold=THRESHOLD [Default: None]

- This is the threshold under which to consider something absent. This should only be used if you have numerical data that should be converted to present or absent based on a threshold. This should be "None" when testing categorical data.

Output:

The G test results are output as tab delimited text, which can be examined in Excel. The output has the following columns:

- OTU: The name of the OTU.
- g_val: The raw test statistic.
- g_prob: The probability that this OTU is non-randomly distributed across the categories.
- Bonferroni_corrected: The probability after correction for multiple comparisons with the Bonferroni correction. In this correction, the p-value is multiplied by the number of comparisons performed (the number of OTUs remaining after applying the filter).
- FDR_corrected: The probability after correction with the “false discovery rate” method. In this method, the raw p-values are ranked from low to high. Each p-value is multiplied by the number of comparisons divided by the rank. This correction is less conservative than the Bonferroni correction. The list of significant OTUs is expected to have the percent of false positives predicted by the p value.
- Contingency table columns: The next columns give the information in the contingency table and will vary in number and name based on the number of categories and their names. The two numbers in brackets represent the number of samples that were observed in those categories and the number that would be expected if the OTU members were randomly distributed across samples in the different categories. These columns can be used to evaluate the nature of a non-random association (e.g. if that OTU is always present in a particular category or if it is never present).
- Consensus lineage: The consensus lineage for that OTU will be listed in the last column if it was present in the input OTU table.

The ANOVA results are output as tab delimited text that can be examined in Excel. The output has the following columns:

- OTU: The name of the OTU.
- prob: The raw probability from the ANOVA test.
- Bonferroni_corrected: The probability after correction for multiple comparisons with the Bonferroni correction. In this correction, the p-value is multiplied by the number of comparisons performed (the number of OTUs remaining after applying the filter).
- FDR_corrected: The probability after correction with the “false discovery rate” method. In this method, the raw p-values are ranked from low to high. Each p-value is multiplied by the number of comparisons divided by the rank. This correction is less conservative than the Bonferroni correction. The list of significant OTUs is expected to have the percent of false positives predicted by the p value.
- Category Mean Columns: Contains one column for each category reporting the mean count of the OTU in that category.
- Consensus lineage: The consensus lineage for that OTU will be listed in the last column if it was present in the input OTU table.

Examples:

If the user would like to perform a G test on their OTU table using default parameters, while testing the category "Sex", they can run the following command:

```
$ python $qdir/otu_category_significance.py -i otu_table.txt -m Mapping_file.txt -s g_test -c Sex
```

If the user would like to perform the same test using numerical qPCR data, where everything below a threshold value should be considered "absent" and everything above that value "present", the user will need to set the threshold by running the following command:

```
$ python $qdir/otu_category_significance.py -i otu_table.txt -m Mapping_file.txt -s g_test -c qPCR -t 0.16
```

Alternatively, the user could run an ANOVA test on the same data by using the following command:

```
$ python $qdir/otu_category_significance.py -i otu_table.txt -m Mapping_file.txt -s ANOVA -c Sex
```

3.9.4 Make OTU Heatmap (Web Application)

QIIME script: ***make_otu_heatmap_html.py***

Once the OTU table has been generated, the user can create an interactive OTU heatmap. This script parses the OTU count table and filters the table by counts per otu (user-specified), then converts the table into a javascript array, which

can be loaded into a web application. This web application allows the user to filter the otu table by number of counts per otu. The user also has the ability to view the table based on taxonomy assignment. Additional features include: the ability to drag rows (up and down) by clicking and dragging on the row headers; and the ability to zoom in on parts of the heatmap by clicking on the counts within the heatmap.

Usage: ***make_otu_heatmap.py*** [options]

Input Arguments:

-i OTU_COUNT_FNAME, --otu_count_fname=OTU_COUNT_FNAME [REQUIRED]

- This is the path to the OTU table (i.e., the resulting OTU table from ***make_otu_table.py*** or ***filter_otu_table.py***).

-n NUM_OTU_HITS, --num_otu_hits=NUM_OTU_HITS [Default: 5]

- This is the minimum number of Samples that an OTU is present in, for an OTU to be kept in the OTU table.

-o DIR_PATH, --dir_path=DIR_PATH [Default:]

- This is the location where the resulting output should be written.

Output:

The interactive heatmap is located in a randomly generated folder where the name of the folder starts with "otu_table...". The resulting folder contains the interactive heatmap (html file) along with a javascript library folder. This web application has been tested in Mozilla Firefox and Safari. Safari is recommended for viewing the OTU Heatmap, since the HTML table generation is much faster.

Examples:

By using the default values ("-n 5), you can then use the code as follows:

```
$ python $qdir/make_otu_heatmap_html.py -i otu_table.txt
```

If you would like to filter the OTU table by a different number of counts per OTU (i.e., 10), you can use the following code:

```
$ python $qdir/make_otu_heatmap_html.py -i otu_table.txt -n 10
```

If you would like to specify a different output directory (i.e., "otu_heatmap"), you can use the following code:

```
$ python $qdir/make_otu_heatmap_html.py -i otu_table.txt -o otu_heatmap
```

3.9.5 Summarize Taxa

QIIME script: ***summarize_taxa.py***

The ***summarize_taxa.py*** script provides summary information of the representation of taxonomic groups within each sample. It takes an OTU table that contains taxonomic information as input. The taxonomic level for which the summary information is provided is designated with the –L option. The meaning of this level will depend on the format of the taxon strings that are returned from the taxonomy assignment step. The taxonomy strings that are most useful are those that standardize the taxonomic level with the depth in the taxonomic strings. For instance, for the RDP classifier taxonomy, Level 2 = Domain (e.g. Bacteria), 3 = Phylum (e.g. Firmicutes), 4 = Class (e.g. Clostridia), 5 = Order (e.g. Clostridiales), 6 = Family (e.g. Clostridiaceae), and 7 = Genus (e.g. Clostridium). By default, the relative abundance of each taxonomic group will be reported, but the raw counts can be returned if –r is set as False.

Usage: ***summarize_taxa.py*** [options]

Input Arguments:

-O OTU_FP, --otu_file=OTU_FP [REQUIRED]

- This is the path to the OTU table (i.e., the resulting OTU table from ***make_otu_table.py*** or ***filter_otu_table.py***).

-o OUT_FP, --output_file=OUT_FP [Default:]

- This is the filename where the results should be written.

-L LEVEL, --level=LEVEL [Default:]

- This is the Level of taxonomy to summarize.

-m CATEGORY_MAPPING, --category_mapping=CATEGORY_MAPPING [Default:]

- This is the user-generated mapping file. The taxon information will be added to the category mapping file and this can be used to color PCoA plots by taxon abundance or to perform statistical tests of taxon/category associations.

-d DELIMITOR, --delimitor=DELIMITOR [default: ;]

- This is the delimiter use to separate taxonomy categories.

-r RELATIVE_ABUNDANCE, --relative_abundance=RELATIVE_ABUNDANCE [Default: True]

- If True, the results report the relative abundance of the lineage in each sample. If False, only the raw counts will be reported.

Output:

There are two possible output formats depending on whether or not a category mapping file is provided with the –m option. If a category mapping file is not provided, a table is returned where the taxonomic groups are each in a row and there is a column for each sample. If a category mapping file is provided, the summary information will be appended to this file. Specifically, a new column will be made for each taxonomic group to which the relative abundances or raw counts will be added to the existing rows for each sample. The addition of the taxonomic information to the category mapping file allows for taxonomic coloration of Principal coordinates plots in the 3d viewer. As described in the ***make_3d_plots.py*** section, principal coordinates plots can be dynamically colored based on any of the metadata columns in the category mapping file. Dynamic coloration of the plots by the relative abundances of each taxonomic group can help to distinguish which taxonomic groups are driving the clustering patterns.

Examples:

The following command can be used to summarize taxa based on the Class, where the default parameters are used (no mapping file, delimiter for RDP ("-d ;") and output relative abundance ("-r True")) and the results are written to the file "Class.txt":

```
$ python $qdir/summarize_taxa.py -O otu_table.txt -L 4 -o Class.txt
```

Optionally the user can have the relative abundances added to the user-generated mapping file, by using the following command:

```
$ python $qdir/summarize_taxa.py -O otu_table.txt -L 4 -m Mapping_file.txt
```

Alternatively, the user may want to output the raw counts of each lineage within a sample, which can be used in the next step for making pie charts, by using the following command:

```
$ python $qdir/summarize_taxa.py -O otu_table.txt -L 4 -r False
```

3.9.6 Make Pie Charts

QIIME script: ***make_pie_charts.py***

This script automates the construction of pie charts showing the breakdown of taxonomy by given levels. The script uses the raw counts file from ***summarize_taxa.py*** to build pie charts. There is also additional functionality that breaks each taxonomic level up by sample. This will create a pie chart for each sample at each specified level.

Usage: ***make_pie_charts.py*** [options]

Input Arguments:

-i COUNTS_FNAME, --input_files=COUNTS_FNAME [REQUIRED]

- This is the text file(s) generated from ***summarize_taxa.py***, where the raw counts for each sample are reported. Multiple files can be passed depending where the different levels are summarized, where each file is separated by commas with no spaces (i.e., "-i Phylum.txt,Class.txt,Genus.txt"

-l LABELS, --labels=LABELS [REQUIRED]

- This is a list of labels for the pie charts corresponding to the input files. They must be in the same order as the input files and separated by commas with no spaces (i.e., "-l Phylum,Class,Genus")

-s, --sample_flag

- If this option is passed, then pie charts will be generated for each of the levels, and it will also create a pie chart for each sample at each level.

-n NUM_CATEGORIES, --num=NUM_CATEGORIES [Default: 20]

- This denotes the number of categories displayed on each pie chart. After this number is reached, the rest of the counts will be put in a single category on the pie chart. The Default is 20 and a number over 40 is not recommended due to the size of the legend.

-o DIR_PATH, --dir-prefix=DIR_PATH [Default:]

- This is the location where the resulting output should be written.

Output:

The script generates an output folder, which contains several files. For each pie chart there is a png and a pdf file. The best way to view all of the pie charts is by opening up the file "taxonomy_summary_pie_chart.html".

Examples:

If you wish to run the code using default parameters, you can use the following command:

```
$ python $qdir/make_pie_charts.py -i Class.txt -l Class
```

If you want specify an output directory (e.g. "pie_charts/", regardless of whether the directory exists, use the following command:

```
$ python $qdir/make_pie_charts.py -i Class.txt -l Class -o pie_charts/
```

Additionally, if you would like to display on a set number of taxa ("-n 10") and generate pie charts for all samples ("-s"), you can use the following command:

```
$ python $qdir/make_pie_charts.py -i Class.txt -l Class -o pie_charts/ -n 10 -s
```

3.9.7 Network Analysis

QIIME script: ***make_otu_network.py***

Network-based analysis is used to display and analyze how OTUs are partitioned between samples. This is a powerful way to display visually large and highly complex datasets in such a way that similarities and differences between samples are emphasized. The visual output of this analysis is a clustering of samples according to their shared OTUs - samples that share more OTUs cluster closer together. The degree to which samples cluster is based on the number of OTUs shared between samples (when OTUs are found in more than one sample) and this is weighted according to the number of sequences within an OTU. In the network diagram, there are two kinds of "nodes" represented, OTU-nodes and sample-nodes. These are shown with symbols such as filled circles and filled squares. If an OTU is found within a sample, the two nodes are connected with a line (an "edge"). (OTUs found only in one sample are given a second, distinct OTU-node shape.) The nodes and edges can then be colored to emphasize certain aspects of the data. For instance, in the initial application of this analysis in a microbial ecology study, the gut bacteria of a variety of mammals was surveyed, and the network diagrams were colored according to the diets of the animals, which highlighted the clustering of hosts by diet category (herbivores, carnivores, omnivores). In a meta-analysis of bacterial surveys across habitat types, the networks were colored in such a way that the phylogenetic classification of the OTUs was highlighted: this revealed the dominance of shared Firmicutes in vertebrate gut samples versus a much higher diversity of phyla represented amongst the OTUs shared by environmental samples.

Not just pretty pictures: the connections within the network are analyzed statistically to provide support for the clustering patterns displayed in the network. A G-test for independence is used to test whether sample-nodes within categories (such as diet group for the animal example used above) are more connected within than a group than expected by chance. Each pair of samples is classified according to whether its members shared at least one OTU, and whether they share a category. Pairs are then tested for independence in these categories (this asks whether pairs that share a category also are equally likely to share an OTU). This statistical test can also provide support for an apparent lack of clustering when it appears that a parameter is not contributing to the clustering.

This OTU-based approach to comparisons between samples provides a counterpoint to the tree-based PCoA graphs derived from the UniFrac analyses. In most studies, the two approaches reveal the same patterns. They can reveal different aspects of the data, however. The network analysis can provide phylogenetic information in a visual manner, whereas PCoA-UniFrac clustering can reveal subclusters that may be obscured in the network. The PCs can be pulled out individually and regressed against other metadata; the network analysis can provide a visual display of shared versus unique OTUs. Thus, together these tools can be used to draw attention to disparate aspects of a dataset, as desired by the author.

In more technical language: OTUs and samples are designated as two types of nodes in a bipartite network in which OTU-nodes are connected via edges to sample-nodes in which their sequences are found. Edge weights are defined as the number of sequences in an OTU. To cluster the OTUs and samples in the network, a stochastic spring-embedded algorithm is used, where nodes act like physical objects that repel each other, and connections act a springs with a spring constant and a resting length: the nodes are organized in a way that minimized forces in the network. These algorithms are implemented in Cytoscape (Shannon et al., 2003).

Usage: ***make_otu_network.py*** [options]

Input Arguments:

-i MAP_FILE, --input_map=MAP_FILE [REQUIRED]

- This is the user-generated mapping file.

-c COUNTS_FILE, --otu_sample_counts=COUNTS_FILE [REQUIRED]

- This is the path to the OTU table (i.e., the resulting OTU table from ***make_otu_table.py*** or ***filter_otu_table.py***).

-o DIR_PATH, --dir-prefix=DIR_PATH

- This is the location where the resulting output should be written.

Output:

The result of ***make_otu_network.py*** consists of a folder containing a props and stats folder, along with several text files.

Examples:

The user can use the following command to create an OTU network, where the results are written to the directory "otu_network/":

```
$ python $qdir/make_otu_network.py -i Mapping_file.txt -c otu_table.txt -o otu_network/
```

Loading Results with Cytoscape:

For this visualization, we will use the real edge table ("real_edge_table.txt") and real node file ("real_node_table").

The following are the direction to use once Cytoscape is installed and open:

1. File -> Import -> Network from Table
    1. Select: "real_edge_table.txt" file
    2. Click: Show Text File Import Options
    3. Click: Transfer first line as attribute names
    4. Select: Source Interaction = Column 1
    5. Select: Target Interaction = Column 2
    6. Click the headers for all other columns to import them (they will turn blue: note, might need to clicked more than once)

- As a result, you should get successful import message.

2. File -> Import -> Attribute from Table
    1. Select: "real_node_table.txt" file
    2. Click: Show Text File Import Options
    3. Click: Transfer first line as attribute name
    4. Click: Import

- As a result, you should get successful import message.

3. Click: VizMapper

- The user can set different properties, e.g. node color, by double-clicking.
    - Select: as "Discrete Mapper"
    - Select: Attribute to color by

- The user can also explore layouts by using Layout from the menu bar.
    - Note: some layouts will take a _very_ long time to run on large datasets

3.10. Within-Sample Diversity Analyses (Including Rarefaction and Curves)

3.10.1 Rarefaction

QIIME script: ***rarefaction.py***

Community ecologists typically describe the diversity of the communities they study. This diversity can assessed within a sample (alpha diversity) or between a collection of samples (beta diversity). Here, we will determine the level of alpha diversity in our samples using a series of scripts from the QIIME pipeline. Before assessing the diversity within a sample for our study, we will rarify our OTU table, which allows for even sampling of sequences. This does not provide curves of diversity by number of sequences in a sample. Rather it creates a series of subsampled OTU tables by random sampling (without replacement) of the input OTU table.

Usage: ***rarefaction.py*** [options]

Input Arguments:

-i INPUT_PATH, --input_path=INPUT_PATH [REQUIRED]

- This is the path to the OTU table (i.e., the resulting OTU table from ***make_otu_table.py*** or ***filter_otu_table.py***).

-o OUTPUT_PATH, --output_path=OUTPUT_PATH [REQUIRED]

- This is the location where the resulting output should be written.

-m MIN, --min=MIN

- This is the minimum number of sequences per sample.

-x MAX, --max=MAX

- This is the maximum number of sequences per samples (inclusive).

-s STEP, --step=STEP

- This is the number of steps that should be used (e.g. range of min, min+step... for level <= max)

-n NUM_REPS, --num-reps=NUM_REPS [Default: 1]

- This is the number of iterations that should be performed for each number of sequences per sample.

-d DEPTH, --depth=DEPTH

- This is the number of sequences per sample, required when generating a single output file. This is primarily used when performing rarefaction in parallel, where each job corresponds to one iteration and step.

--small_included [default: False]

- If this argument is passed, samples containing fewer sequences than the user-defined level are included in the output and not rarefied.

Output:

The results of ***rarefaction.py*** consist of a number of files, which depend on the minimum/maximum number of sequences per samples, steps and iterations. The files have the same otu table format as the input otu_table.txt, and are named in the following way: rarefaction_100_0.txt, where "100" corresponds to the sequences per sample and "0" for the iteration.

Example:

Single file rarefaction:

If the user would like to produce a single rarefied OTU table, they can set the minimum ("-m") and maximum ("-x") number of sequences per sample to be equal (i.e. 400), while using a step ("-s") of 1 and performing only 1 iteration ("-n"), they can use the following command:

```
$ python $qdir/rarefaction.py -i otu_table.txt -m 400 -x 400 -s 1 -n 1
```

Multiple file (batch) rarefaction:

An example of this script, where the user sets the minimum ("-m") and maximum ("-x") number of sequences per sample to 100 and 1200, respectively, while using steps ("-s") of 100, performing 2 iterations ("-n") and outputting the results to the directory "rarefaction_tables/" is shown by the following command:

```
$ python $qdir/rarefaction.py otu_table.txt -m 100 -x 1200 -s 100 -n 2 -o rarefaction_tables/
```

As a result, this command produces subsamples of the input "otu_table.txt" at 100 seqs per sample (twice), 200 seqs per sample (twice) ... 1200 seqs per sample (twice), which produces 24 files total to the "rarefaction_tables" directory.

By default, any sample containing fewer sequences in the input file than the requested number of sequences per sample is removed from the output rarefied otu table. To include samples with fewer than the requested number, you can use the following command:

```
$ python $qdir/rarefaction.py otu_table.txt -m 100 -x 1200 -s 100 -n 2 -o rarefaction_tables/ --small_included
```

3.10.2 Alpha-Diversity

QIIME script: ***alpha_diversity.py***

The rarefaction tables are the basis for calculating diversity metrics, which reflect the diversity within the sample based on taxon counts of phylogeny. The QIIME pipeline allows users to conveniently calculate more than two dozen different diversity metrics. The full list of available metrics is available by passing the option -s to the script ***alpha_diversity.py***. Every metric has different strengths and limitations - technical discussion of each metric is readily available online and in ecology textbooks, but is beyond the scope of this document.

Usage: ***alpha_diversity.py*** [options]

Input Arguments:

-i INPUT_PATH, --input_path=INPUT_PATH [REQUIRED]

- This is the path to the OTU table (i.e., the resulting OTU table from ***make_otu_table.py***, ***filter_otu_table.py*** or ***rarefaction.py***). If you are using the results from ***rarefaction.py***, where there are multiple rarefied tables, then you can pass the directory name to allow for batch processing.

-o OUTPUT_PATH, --output_path=OUTPUT_PATH [REQUIRED]

- This is the filename (for single file input) or the path location to where the results should be written if a directory is input.

-m METRICS, --metrics=METRICS [REQUIRED]

- These are the metrics to perform. If more than one metric is given, then they should be comma delimited.

-s, --show_metrics

- If this argument is passed, then a list of available alpha diversity metrics will be displayed without running the actual script. A list of acceptable metrics are shown below.

-t TREE_PATH, --tree_path=TREE_PATH [Default: none]

- This is the path to the newick formatted tree file (i.e., the resulting tree from ***make_phylogeny.py***). This argument is only required for metrics which are phylogenetically based.

Non-phylogeny based metrics:

- berger_parker_d
- brillouin_d
- chao1
- chao1_confidence
- dominance
- doubles
- equitability
- fisher_alpha
- heip_e

- kempton_taylor_q
- margalef
- mcintosh_d
- mcintosh_e
- menhinick
- michaelis_menten_fit
- observed_species
- osd
- reciprocal_simpson
- robbins
- shannon
- simpson
- simpson_e
- singles
- strong

Phylogeny based metrics:

- PD_whole_tree

Output:

Depending on whether the user passed a single file or multiple files to the script, the resulting output will be the same number of files as supplied by the user. The resulting file(s) is a tab-delimited text file, where the columns correspond to alpha diversity metrics and the rows correspond to samples and their calculated diversity measurements. When multiple files are given, script processes every file in the given folder, and creates a file "alpha_..." in the output directory.

Example Output:

```
          simpson       PD_whole_tree    observed_species
PC.354    0.925         2.83739          16.0
PC.355    0.915         3.06609          14.0
PC.356    0.945         3.10489          19.0
PC.481    0.945         3.65695          19.0
PC.593    0.91          3.3776           15.0
PC.607    0.92          4.13397          16.0
PC.634    0.9           3.71369          14.0
PC.635    0.94          4.20239          18.0
PC.636    0.925         3.78882          16.0
```

Examples:

Single file alpha diversity:

To perform alpha diversity (e.g. chao1) on a single OTU table, where the results are output to "alpha_div.txt", you can use the following command:

```
$ python $qdir/alpha_diversity.py -i otu_table.txt -m chao1 -o alpha_div.txt
```

Note: Since this is a non-phylogenetic metric, the tree does not need to be supplied.

In the case that you would like to perform alpha diversity using a phylogenetic metric (e.g. PD_whole_tree), you can use the following command:

```
$ python $qdir/alpha_diversity.py -i otu_table.txt -m PD_whole_tree -o alpha_div.txt -t repr_set.tre
```

If you would like to run multiple metrics at once (comma-separated), you can use the following command:

```
$ python $qdir/alpha_diversity.py -i otu_table.txt -m chao1,PD_whole_tree -o alpha_div.txt -t repr_set.tre
```

Multiple file (batch) alpha diversity:

To perform alpha diversity on multiple OTU tables (resulting files from ***rarefaction.py***), the only change from the command above is that instead of outputting the results to single file, the user should specify an output directory (e.g. alpha_div_chao1_PD/) as shown by the following command:

```
$ python $qdir/alpha_diversity.py -i rarefaction_tables/ -m chao1,PD_whole_tree -o alpha_div_chao1_PD/ -t
        repr_set.tre
```

3.10.3 Collate Alpha Diversity Results

QIIME script: ***collate_alpha.py***

When performing batch analyses on the OTU table (e.g. rarefaction followed by alpha diversity), the result of ***alpha_diversity.py*** comprises many files, which need to be concatenated into a single file for generating rarefaction curves. This script takes the resulting files from batch alpha diversity and collates them into a single (one file for each metric used).

This script transforms a series of files, named (e.g. alpha_rarefaction_20_0.txt, alpha_rarefaction_20_1.txt,...) into a (usually much smaller) set of files named (e.g. chao1.txt, PD_whole_tree.txt, etc...), where the columns correspond to samples and rows to the rarefaction files inputted, as shown by the following:

	sequences per sample	iteration	PC.354	PC.355
alpha_rarefaction_20_0.txt	20	0	0.925	0.915
alpha_rarefaction_20_1.txt	20	1	0.9	0.89
alpha_rarefaction_20_2.txt	20	2	0.88	0.915
alpha_rarefaction_20_3.txt	20	3	0.91	0.93
...				

Usage: ***collate_alpha.py*** [options]

Input Arguments:

-i INPUT_PATH, --input_path=INPUT_PATH [REQUIRED]

- This is the location where the batch alpha diversity files are located (a directory).

-o OUTPUT_PATH, --output_path=OUTPUT_PATH [REQUIRED]

- This is the location where the resulting output should be written.

-e EXAMPLE_PATH, --example_path=EXAMPLE_PATH [default: chosen automatically (see usage string)]

- This is an example alpha_diversity analysis file, containing all samples and all metrics to be included in the collated result.

Example:

The user inputs the results from batch alpha diversity (e.g. alpha_div_chao1_PD/) and the location where the results should be written (e.g. alpha_collated/), as shown by the following command:

```
$ python $qdir/collate_alpha.py -i alpha_div_chao1_PD/ -o alpha_collated/
```

3.10.4 Rarefaction Curves

QIIME script: ***make_rarefaction_plots.py***

Once the batch alpha diversity files have been collated, you may want to compare the diversity using plots. Using the results from ***collate_alpha.py***, you can plot the samples and or by category in the mapping file using this script.

Usage: ***make_rarefaction_plots.py*** [options]

Input Arguments:

-m MAP, --map=MAP [REQUIRED]

- This is the user-generated mapping file.

-r RAREFACTION, --rarefaction=RAREFACTION [REQUIRED]

- This is the name of the rarefaction file (i.e., one of the resulting txt files from ***collate_alpha.py***).

-p PREFS, --prefs=PREFS [Default: ALL]

- These are the categories for which plots should be made, from the user-generated mapping file. The category must match the name of a column header in the mapping file exactly. Multiple categories can be list as long as they are comma separated without spaces. If plots should be made for all categories, use the word "ALL".

-i IMAGETYPE, --imagetype=IMAGETYPE [Default: png]

- This is the file format for the plots generated. Valid choices are: jpg, gif, png, svg, and pdf.

-d RESOLUTION, --resolution=RESOLUTION [Default: 75]

- This is the resolution (in d.p.i.) for each generated plot.

-o DIR_PATH, --dir_path=DIR_PATH [Default: .]

- This is the location where the resulting output should be written.

Output:

The result of this script produces a folder and within that folder there are sub-folders for each data file (metric) supplied as input. Within the sub-folders, there will be images for each of the categories specified by the user.

Examples:

For generated rarefaction plots using the default parameters, including the mapping file and one rarefaction file, you can use the following command:

```
$ python $qdir/make_rarefaction_plots.py -m Mapping_file.txt -r chao1.txt
```

If you would like to generate plots for multiple files, you can use the following command:

```
$ python $qdir/make_rarefaction_plots.py -m Mapping_file.txt -r chao1.txt,PD_whole_tree.txt
```

In the case that you want to make plots for a specific category (i.e., pH), you can use the following command:

```
$ python $qdir/make_rarefaction_plots.py -m Mapping_file.txt -r chao1.txt -p pH
```

Optionally, you can change the resolution ("-d") and the type of image created ("-i"), by using the following command:

```
$ python $qdir/make_rarefaction_plots.py -m Mapping_file.txt -r chao1.txt -p pH -d 180 -i pdf
```

3.11. Between-Sample Diversity Analyses (OTU-based and Phylogenetic)

3.11.1 Determine Between-Sample (Beta-Diversity) Metrics

QIIME script: ***beta_diversity.py***

This segment utilizes the script ***beta_diversity.py***, where the input for this script is the OTU table containing the number of sequences observed in each OTU (rows) for each sample (columns). For more information pertaining to the OTU table refer to the section using ***make_otu_table.py***. If the user would like phylogenetic beta diversity metrics using UniFrac, a phylogenetic tree must also be passed as input (see ***make_phylogeny.py***). The output of this script is a distance matrix containing a dissimilarity value for each pairwise comparison.

A number of metrics are currently supported, including unweighted and weighted UniFrac (pass the -s option to ***beta_diversity.py***). In general, because unifrac uses phylogenetic information, one of the unifrac metrics is recommended, as results can be vastly more useful (Hamady & Knight, 2009). Weighted unifrac is sensitive to factors affecting the relative abundance of different taxa, such as salinity or pH while unweighted unifrac considers only the presence or absence of taxa, and is sensitive to factors such as nutrient availability (Costello, personal communication). Typically both weighted and unweighted unifrac are applied. The complete list is as follows:

Non-phylogenetic beta diversity metrics. These are count based metrics which is based on the OTU table:

- binary_dist_chisq: Binary chi-square distance
- binary_dist_chord: Binary chord distance
- binary_dist_euclidean: Binary euclidean distance
- binary_dist_hamming: Binary Hamming distance (binary Manhattan distance)
- binary_dist_jaccard: Binary Jaccard distance (binary Soergel distance)
- binary_dist_lennon: Binary Lennon distance
- binary_dist_ochiai: Binary Ochiai distance
- binary_dist_pearson: Binary Pearson distance
- binary_dist_sorensen_dice: Binary Sörensen-Dice distance (binary Bray-Curtis distance or binary Whittaker distance)
- dist_bray_curtis: Bray-Curtis distance (normalized Manhattan distance)
- dist_canberra: Canberra distance
- dist_chisq: Chi-square distance
- dist_chord: Chord distance
- dist_euclidean: Euclidean distance
- dist_gower: Gower distance
- dist_hellinger: Hellinger distance
- dist_kulczynski: Kulczynski distance
- dist_manhattan: Manhattan distance
- dist_morisita_horn: Morisita-Horn distance
- dist_pearson: Pearson distance
- dist_soergel: Soergel distance
- dist_spearman_approx: Spearman rank distance
- dist_specprof: Species profile distance

Phylogenetic beta diversity metrics. These metrics are based on UniFrac, which takes into account the evolutionary relationship between sequences:

- dist_unifrac_G: The G metric calculates the fraction branch length in the sample i + sample j tree that is exclusive to sample i and it is asymmetric.
- dist_unifrac_G_full_tree: The full_tree version calculates the fraction of branch length in the full tree that is exclusive to sample i and it is asymmetric.
- dist_unweighted_unifrac: This is the standard unweighted UniFrac, which is used to assess 'who's there' without taking in account the relative abundance of identical sequences.
- dist_unweighted_unifrac_full_tree: Typically, when computing the dissimilarity between two samples, unifrac considers only the parts of the phylogenetic tree contained by otus in either sample. The full_tree modification considers the entire supplied tree.
- dist_weighted_normalized_unifrac: Weighted UniFrac with normalized values and is used to include abundance information. The normalization adjusts for varying root-to-tip distances.
- dist_weighted_unifrac: Weighted UniFrac.

Usage: ***beta_diversity.py*** [options]

Input Arguments:

-i INPUT_PATH, --input_path=INPUT_PATH [REQUIRED]

- This is the path to the OTU table (i.e., the resulting OTU table from ***make_otu_table.py***, ***filter_otu_table.py*** or ***rarefaction.py***). If you are using the results from ***rarefaction.py***, where there are multiple rarefied tables, then you can pass the directory name to allow for batch processing.

-o OUTPUT_DIR, --output_dir=OUTPUT_DIR [REQUIRED]

- This is the location where the resulting output should be written.

-m METRICS, --metrics=METRICS

- These are the metrics to perform.

-s, --show_metrics

- If this argument is passed, then a list of available beta diversity metrics will be displayed without running the actual script. A list of acceptable metrics are shown below.

-t TREE_PATH, --tree_path=TREE_PATH [Default: none]

- This is the path to the newick formatted tree file (i.e., the resulting tree from make_phylogeny.py). This argument is only required for metrics which are phylogenetically based.

Output:

The output of ***beta_diversity.py*** is a folder containing text files, each a distance matrix between samples.

Examples:

Single file beta diversity:

To perform beta diversity (e.g. dist_euclidean) on a single OTU table, where the results are output to "beta_div.txt", you can use the following command:

```
$ python $qdir/beta_diversity.py -i otu_table.txt -m dist_euclidean -o beta_div/
```

Note: Since this is a non-phylogenetic metric, the tree does not need to be supplied.

In the case that you would like to perform beta diversity using a phylogenetic metric (e.g. weighted_unifrac), you can use the following command:

```
$ python $qdir/beta_diversity.py -i otu_table.txt -m weighted_unifrac -o beta_div/ -t repr_set.tre
```

Multiple file (batch) beta diversity:

To perform beta diversity on multiple OTU tables (resulting files from ***rarefaction.py***), the only change from the command above is that instead of outputting the results to single file, the user should specify an output directory (e.g. beta_div_weighted_unifrac/) as shown by the following command:

```
$ python $qdir/beta_diversity.py -i rarefaction_tables/ -m weighted_unifrac -o beta_div/ -t repr_set.tre
```

3.11.2 Distance Histograms (Web Application)

QIIME script: ***make_distance_histograms.py***

To visualize the distance between samples and/or categories in the mapping file, the user can generate histograms to

represent the distances between samples. This script generates an HTML file, where the user can compare the distances between samples based on the different categories associated to each sample in the mapping file.

Usage: ***make_distance_histograms.py*** [options]

Input Arguments:

-d DISTANCE_MATRIX_FILE, --distance_matrix_file=DISTANCE_MATRIX_FILE [REQUIRED]

- This is the path to the distance matrix file (i.e., resulting file from ***beta_diversity.py***).

-m MAPPING_FILE, --mapping_file=MAPPING_FILE [REQUIRED]

- This is the user-generated mapping file.

-p PREFS_FILE, --prefs_file=PREFS_FILE [Default: none]

- This is the user-generated preferences file. NOTE: This is a file with a dictionary containing preferences for the analysis. This dict must have a "Fields" key mapping to a list of desired fields.

-o DIR_PATH, --dir_path=DIR_PATH [Default: .]

- This is the location where the resulting output should be written.

--fields=FIELDS [REQUIRED]

- This is the categories to plot from the user-generated mapping file. The categories must match the name of a column header in the mapping file exactly and multiple categories can be list by comma separating them without spaces. This overwrites "Fields" in preferences file.

--monte_carlo [Default: False]

- If this argument is passed, then Monte Carlo will be performed on the distances.

--html_output [Default: False]

- If this argument is passed, then the output will be written in HTML format.

Output:

The result of this script will be a folder containing images and/or an html file (with appropriate javascript files), depending on the user-defined parameters.

Examples:

In the following command, the user only supplies a distance matrix (i.e. resulting file from ***beta_diversity.py***), the user-generated mapping file and one category (e.g. pH):

```
$ python $qdir/make_distance_histograms.py -d beta_div.txt -m Mapping_file.txt --fields pH
```

For comparison of multiple categories (e.g. pH, salinity), you can use the following command:

```
$ python $qdir/make_distance_histograms.py -d beta_div.txt -m Mapping_file.txt --fields pH,salinity
```

If the user would like to write the result to a dynamic HTML, you can use the following command:

```
$ python $qdir/make_distance_histograms.py -d beta_div.txt -m Mapping_file.txt --fields pH --html_output
```

In the case that the user generates their own preferences file (prefs.txt), they can use the following command:

$ python $qdir/make_distance_histograms.py -d beta_div.txt -m Mapping_file.txt -p prefs.txt

Note: In the case that a preferences file is passed, the user does not need to supply fields in the command-line.

3.11.3 Principal Coordinates Analysis (PCoA)

QIIME script: ***principal_coordinates.py***

Principal Coordinate Analysis (PCoA) is commonly used to compare groups of samples based on phylogenetic or count-based distance metrics (see section on ***beta_diversity.py***).

Usage: ***principal_coordinates.py*** [options]

Input Arguments:

-i INPUT_PATH, --input_path=INPUT_PATH [REQUIRED]

- This is the path to the distance matrix file (i.e., resulting file from ***beta_diversity.py***).

-o OUTPUT_PATH, --output_path=OUTPUT_PATH [REQUIRED]

- This is the filename where the results should be written.

Output:

The resulting output file consists of each component (columns) along with the loading for each sample (rows). Pairs of components can then be graphed to view the relationships between samples. The bottom of the output file contains the eigenvalues and % variation explained for each component.

Example:

For this script, the user supplies a distance matrix (i.e., resulting file from ***beta_diversity.py***), along with the output filename (e.g. "beta_div_coords.txt"), as follows:

$ python $qdir/principal_coordinates.py -i beta_div.txt -o beta_div_coords.txt

3.11.4 Two-Dimensional (2D) PCoA Plots

QIIME script: ***make_2d_plots.py***

This script automates the construction of 2D plots from the first three columns of the PCoA output file generated in **Section 3.10.3** (e.g. P1 vs. P2, P2 vs. P3, etc., where P1 is the first component). Components are ordered based on the amount of variance explained (i.e. P1 explains the largest proportion of the variance across the entire dataset). Typically, samples are graphed along the first few components (P1 v. P2, P1 v. P3, and P2 v. P3). Groups of samples that are close to each other along a given axis can be considered to be more similar to each other with respect to the variance explained by that component. By overlaying colors corresponding to important categories in the mapping file (e.g. diet, health/disease status, sampling location, etc.), it is possible to determine if samples cluster according to a measured parameter; however, additional statistical tests must be performed to determine if the observed clustering is significant (e.g. **Section 3.10.4**). All plots could also be constructed manually in a user-specified program.

Usage: ***make_2d_plots.py*** [options]

Input Arguments:

-i COORD_FNAME, --coord_fname=COORD_FNAME [REQUIRED]

- This is the path to the principal coordinates file (i.e., resulting file from ***principal_coordinates.py***).

-m MAP_FNAME, --map_fname=MAP_FNAME [Default: none]

- This is the user-generated mapping file.

-b COLORBY, --colorby=COLORBY [Default: none]

- This is the categories to color by in the plots from the user-generated mapping file. The categories must match the name of a column header in the mapping file exactly and multiple categories can be list by comma separating them without spaces. The user can also combine columns in the mapping file by separating the categories by "&&" without spaces.

-o DIR_PATH, --dir_path=DIR_PATH [Default:]

- This is the location where the resulting output should be written.

Output:

This script generates an output folder, which contains several files. To best view the 2D plots, it is recommended that the user views the *_pca_2D.html file.

Examples:

If you just want to use the default output, you can supply the principal coordinates file (i.e., resulting file from ***principal_coordinates.py***), where the default coloring will be based on the SampleID as follows:

```
$ python $qdir/make_2d_plots.py -i beta_div_coords.txt
```

If you want to give an specific output directory (e.g. "2d_plots"), use the following code:

```
$ python $qdir/make_2d_plots.py -i principal_coordinates-output_file -o 2d_plots/
```

Additionally, the user can supply their mapping file ("-m") and a specific category to color by ("-b") or any combination of categories. When using the -b option, the user can specify the coloring for multiple mapping labels, where each mapping label is separated by a comma, for example: -b 'mapping_column1,mapping_column2'. The user can also combine mapping labels and color by the combined label that is created by inserting an '&&' between the input columns, for example: -b 'mapping_column1&&mapping_column2'.

If the user wants to color by specific mapping labels, they can use the following code:

```
$ python $qdir/make_2d_plots.py -i beta_div_coords.txt -m Mapping_file.txt -b 'mapping_column'
```

or use some of the suggestions from above:

```
$ python $qdir/make_2d_plots.py -i beta_div_coords.txt -m Mapping_file.txt –b
        'mapping_column1,mapping_column1&&mapping_column2'
```

As an alternative, the user can supply a preferences (prefs) file, using the -p option. The prefs file allows the user to give specific samples their own columns within a given mapping column. This file also allows the user to perform a color gradient, given a specific mapping column. An overview and an example preferences file is shown below:

This shows a generic overview of a preferences file (.txt):

```
--------------------Beginning of Generic Prefs File ----------------------------------

{
    'Name_for_color_scheme1':
    {
        'column':'mapping_column1',
        'colors':{'Sample1':'red','Sample2':'blue'},
    },
    'Name_for_color_scheme2':
```

```
     {
         'column':'mapping_column2',
         'colors':(('red',(0,100,100)),('blue',(240,100,100)))
     }
}

-----------------------End of Generic Prefs File -------------------------------------
```

This shows an example of a Prefs file (.txt):

```
--------------------Beginning of Example Prefs File ----------------------------------

{
    'Color_By_Hand':
    {
        'column':'Hand',
        'colors':{'Left':'red','Right':'blue'},
    },
    'Color_By_pH_gradient':
    {
        'column':'pH',
        'colors':(('red',(0,100,100)),('blue',(240,100,100)))
    }
}

-----------------------End of Example Prefs File -------------------------------------
```

If the user wants to color by using the preferences file (e.g. prefs.txt), they can use the following code:

```
$ python $qdir/make_2d_plots.py beta_div_coords.txt -m Mapping_file.txt -p prefs.txt
```

3.11.5 Three-Dimensional (3D) PCoA Plots

QIIME script: ***make_3d_plots.py***

This script automates the construction of 3D plots (kinemage format) from the PCoA output file generated by ***principal_coordinates.py*** (e.g. P1 vs. P2 vs. P3, P2 vs. P3 vs. P4, etc., where P1 is the first component). All plots could also be constructed manually in a user-specified program.

Usage: ***make_3d_plots.py*** [options]

Input Arguments:

-i COORD_FNAME, --coord_fname=COORD_FNAME [REQUIRED]

- This is the path to the principal coordinates file (i.e., resulting file from ***principal_coordinates.py***).

-m MAP_FNAME, --map_fname=MAP_FNAME [Default: none]

- This is the user-generated mapping file.

-b COLORBY, --colorby=COLORBY [Default: none]

- This is the categories to color by in the plots from the user-generated mapping file. The categories must match the name of a column header in the mapping file exactly and multiple categories can be list by comma separating them without spaces. The user can also combine columns in the mapping file by separating the categories by "&&" without spaces.

-a CUSTOM_AXES, --custom_axes=CUSTOM_AXES [Default: none]

- This is the category from the user-generated mapping file to use as a custom axis in the plot. For instance,

there is a pH category and would like to seethe samples plotted on that axis instead of PC1, PC2, etc., one can use this option. It is also useful for plotting time-series data.

-p PREFS_PATH, --prefs_path=PREFS_PATH [Default: none]

- This is the user-generated preferences file. NOTE: This is a file with a dictionary containing preferences for the analysis.

-o DIR_PATH, --dir_path=DIR_PATH [Default:]

- This is the location where the resulting output should be written.

Output:

By default, the script will plot the first three dimensions in your file. Other combinations can be viewed using the "Views:Choose viewing axes" option in the KiNG viewer (Chen, Davis, & Richardson, 2009), which may require the installation of kinemage software. The first 10 components can be viewed using "Views:Paralled coordinates" option or typing "/". The mouse can be used to modify display parameters, to click and rotate the viewing axes, to select specific points (clicking on a point shows the sample identity in the low left corner), or to select different analyses (upper right window). Although samples are most easily viewed in 2D, the third dimension is indicated by coloring each sample (dot/label) along a gradient corresponding to the depth along the third component (bright colors indicate points close to the viewer).

Examples:

If you just want to use the default output, you can supply the principal coordinates file (i.e., resulting file from ***principal_coordinates.py***), where the default coloring will be based on the SampleID as follows:

```
$ python $qdir/make_3d_plots.py -i beta_div_coords.txt
```

If you want to give an specific output directory (e.g. "3d_plots"), use the following code:

```
$ python $qdir/make_3d_plots.py -i principal_coordinates-output_file -o 3d_plots/
```

Additionally, the user can supply their mapping file ("-m") and a specific category to color by ("-b") or any combination of categories. When using the -b option, the user can specify the coloring for multiple mapping labels, where each mapping label is separated by a comma, for example: -b 'mapping_column1,mapping_column2'. The user can also combine mapping labels and color by the combined label that is created by inserting an '&&' between the input columns, for example: -b 'mapping_column1&&mapping_column2'.

If the user wants to color by specific mapping labels, they can use the following code:

```
$ python $qdir/make_3d_plots.py -i beta_div_coords.txt -m Mapping_file.txt -b 'mapping_column'
```

or use some of the suggestions from above:

```
$ python $qdir/make_3d_plots.py -i beta_div_coords.txt -m Mapping_file.txt -b
        'mapping_column1,mapping_column1&&mapping_column2'
```

As an alternative, the user can supply a preferences (prefs) file, using the -p option. The prefs file allows the user to give specific samples their own columns within a given mapping column. This file also allows the user to perform a color gradient, given a specific mapping column. An overview and an example prefs file is shown above in "Two-Dimensional (2D) PCoA Plots".

If the user wants to color by using the prefs file (e.g. prefs.txt), they can use the following code:

```
$ python $qdir/make_3d_plots.py -i beta_div_coords.txt -m Mapping_file.txt -p prefs.txt
```

3.12 Build UPGMA Tree Comparing Samples

QIIME script: ***hierarchical_cluster.py***

In addition to using PCoA, it can be useful to cluster samples using UPGMA (Unweighted Pair Group Method with Arithmetic mean, also known as average linkage). As with PCoA, the input to this step is a distance matrix (i.e. resulting file from ***beta_diversity.py***).

Usage: ***hierarchical_cluster.py*** [options]

Input Arguments:

-i INPUT_PATH, --input_path=INPUT_PATH [REQUIRED]

- This is the path to the distance matrix file (i.e., resulting file from ***beta_diversity.py***). If beta diversity was performed following rarefaction (batch), a directory can be passed instead of a single file.

-o OUTPUT_PATH, --output_path=OUTPUT_PATH [REQUIRED]

- This is the location where the resulting output should be written or the filename for a single file input.

Output:

The output is a newick formatted tree compatible with most standard tree viewing programs. Batch processing is also available, allowing the analysis of an entire directory of distance matrices.

Examples:

Hierarchical Cluster (Single File):

If the user is performing hierarchical clustering on a single file (i.e. resulting file from ***beta_diversity.py***) and defining the output filename (e.g. beta_div_cluster.tre), they can use the following command:

```
$ python $qdir/hierarchical_cluster.py -i beta_div.txt -o beta_div_cluster.tre
```

Hierarchical Cluster (Multiple Files):

If the user is performing hierarchical clustering on a directory (i.e. resulting directory from ***beta_diversity.py***) and defining the output directory (e.g. beta_div_weighted_clusters/), they can use the following command:

```
$ python $qdir/hierarchical_cluster.py -i beta_div_weighted_unifrac/ -o beta_div_weighted_clusters/
```

3.13 Build UPGMA Trees from Jackknifed Samples to Assign Confidence Values to UPGMA Nodes

Although the analyses in the previous sections allow for clustering and comparison between groups of samples, they do not directly measure the robustness of individual clusters. One technique for doing this is called jackknifing (repeatedly resampling a subset of the available data from each sample). This process will utilize a new script, ***tree_compare.py***, along with scripts from previous sections (***rarefaction.py***, ***beta_diversity.py*** and ***hierarchical_cluster.py***).

To perform jackknifing, you should perform the following steps first:

1. Perform rarefaction to resample a subset of each dataset (i.e. resulting file from ***make_otu_table.py***), where the minimum and maximum number of sequences per sample are equal (e.g. 700 for both), with 1 step and multiple iterations (e.g. 100), where the results are written to a directory (e.g. "jackknife_rare/"). You must include all small samples ("--small_included"), because the trees must contain equivalent tips for comparison. You can perform this step using the following command:

```
$ python $qdir/rarefaction.py -i otu_table.txt -m 700 -x 700 -s 1 -n 100 -o jackknife_rare/ --small_included
```

2. Perform beta diversity on the rarefied tables, where the input is the output directory from the previous step and the distance matrix tables produced by beta diversity are output to a directory (e.g. "jackknife_beta_div/"). The user must also specify a metric (e.g. "dist_weighted_unifrac") to use and in the case the metric is phylogenetic a tree (i.e. resulting file from ***make_phylogeny.py***) must also be provided, as shown by the following command:

```
$ python $qdir/beta_diversity.py -i jackknife_rare/ -o jackknife_beta_div/ -m dist_weighted_unifrac
        -t repr_set.tre
```

3. Create UPGMA trees for each distance matrix created in the previous step. The input for generating the UPGMA trees is the output directory from the previous beta diversity step (e.g. "jackknife_beta_div/") and the resulting trees will be written in to the output directory (e.g. "jackknife_upgma_trees/"), as shown by the following command:

```
$ python $qdir/hierarchical_cluster.py -i jackknife_beta_div -o jackknife_upgma_trees/
```

4. Finally, we will compare the resampled UPGMA trees to our tree constructed from the entire dataset (i.e., resulting file from ***make_phylogeny.py***):

   QIIME script: ***tree_compare.py***

   Usage: ***tree_compare.py*** [options]

   Input Arguments:

   -m MASTER_TREE, --master_tree=MASTER_TREE [REQUIRED]

   - This is the path to the newick formatted tree file (i.e., the resulting tree from ***make_phylogeny.py***).

   -s SUPPORT_DIR, --support_dir=SUPPORT_DIR [REQUIRED]

   - This is the path to the directory containing the jackknife supported trees (i.e., resulting directory from batch ***hierarchical_cluster.py***)

   -o OUTPUT_DIR, --output_dir=OUTPUT_DIR [REQUIRED]

   - This is the location where the resulting output should be written.

   Output:

   The result of ***tree_compare.py*** contains the master tree, now with internal nodes uniquely named, a separate bootstrap/jackknife support file, listing the support for each internal node, and a “jackknife_named_nodes.tre” tree, for use with tree visualization software (e.g. FigTree).

   Example:

   Given the sample upgma tree generated by the user for the entire dataset, the directory of jackknife supported trees (i.e., the resulting directory from ***hierarchical_cluster.py***) and the directory to write the results for the tree comparisons, you can use the following command:

```
$ python $qdir/tree_compare.py -m sample_clustering.tre -s jackknife_upgma_trees/ -o
        jackknife_comparison/
```

   Additionally, ***tree_compare.py*** can be used to compare any set of support trees to a master tree, making it easily expandable to additional algorithms.

4. Special-Purpose Tutorials

4.1 Parallel Runs

QIIME supports running several of its slower steps in parallel in a cluster (or other multiple processor/core) environment. Currently, these include:

- Assignment of taxonomy with BLAST, via Qiime/qiime/parallel/assign_taxonomy_blast.py
- Assignment of taxonomy with RDP, via Qiime/qiime/parallel/assign_taxonomy_rdp.py
- Sequence alignment with PyNAST, via Qiime/qiime/parallel/align_seqs_pynast.py

4.1.1 Writing a cluster_jobs Script Specific to your Cluster environment

To make QIIME parallelization useful in different computing environments users are required to provide a script which can start jobs on their system, referred to here as a 'cluster_jobs' script. The cluster_jobs script takes as its two parameterts: (1) a single file which lists the commands to be run (referred to as a 'jobs_list' file), with one command per line; and (2) a string to use as a prefix when constructing unique job identifiers.

The lines in an example jobs_list file might be:

```
python pick_otus.py –i inseqs_file1.fasta
python pick_otus.py –i inseqs_file2.fasta
python pick_otus.py –i inseqs_file3.fasta
```

If passed to your cluster_jobs script, this should start three separate jobs corresponding to each of the commands.

The call to the cluster_jobs script in QIIME's parallel scripts looks like the following:

```
$ CLUSTER_JOBS_FP -ms job_list.txt JOB_ID
```

where CLUSTER_JOBS_FP is the path to your cluster_jobs script and is passed to the parallel scripts via the -U parameter. JOB_ID is intended to be used as a prefix by the cluster_jobs script when creating a unique identifier for each job (and will be passed to the parallel scripts via -X). The same JOB_ID is also used by the Qiime parallel scripts when creating names for temporary files and directories. The -ms indicates that the job files should be made (-m) and submitted (-s).

Once you have written a cluster_jobs script for your specific environment that can be called via the above interface, running QIIME jobs in parallel should be straight-forward. The parallel variants of the scripts use the same parameters as the serial versions of the scripts, with some additional options in the parallel scripts. Options -N through -Z (capital N through capital Z) are reserved in QIIME for parallel scripts, and in most cases the defaults can be defined in your qiime_config file (see **Section 4.2**).

4.1.2 Example Run of PyNAST in Parallel

The following command will start a parallel run of PyNAST, which uses the same interface as the ***align_seqs.py*** script, where the results are written the an output directory "parallel_aligned_seqs/":

```
$ python $qdir/parallel/align_seqs_pynast.py -i repr_set_seqs.fasta -t /ref_set_seqs.fasta -o /home/caporaso/out
```

The important thing to note is that this command is that same that would be used to call serial (single processor) PyNAST, except that instead of calling $qdir/parallel/align_seqs_pynast.py, you would call ***align_seqs.py*** to start the run on a single processor. The output from this parallel run is the same as the output would be from the serial run.

4.1.3 Details of the Parallelization

This section provides some information on details of the parallelization which are hidden from the user, but provided for users who are interested in what is happening behind-the-scenes.

The parallelization works as follows. First, the input file (-i) is split into JOBS_TO_START (-O) different roughly equal-sized files. The serial version of the script -- ***align_seqs.py*** -- is then called on each of these split files as a separate job. Each of these jobs therefore writes its own output files (alignment, log, and failure files). One additional job, the poller, is started to monitor each of the jobs via their output files. When all expected output files exist, the poller will merge the individual output files and clean up any temporary files including the output files created by each of the individual runs. Cleaning up temporary files can be suppressed by passing -R, which is useful for debugging. Bypassing the polling

system all-together can be achieved by passing -W.

4.2 The qiime_config File

First things first: you should not edit or remove Qiime/qiime_config.

Some QIIME scripts, at this stage primarily the parallel scripts, read default values from a qiime_config file. The default location of this file is in your top-level QIIME directory (Qime/qiime_config). QIIME scripts pull default values from this file which are system-specific, such as paths to executable files, and these can be overwritten for convenience. The recommended procedure for overwriting these defaults is to copy the qiime_config file to either ~/.qiime_config or a location specified by the environment variable $QIIME_CONFIG_FP.

The Qiime configuration values should only be modified in these copies of the qiime_config file, as changes to the Qiime/qiime_config version may be overwritten in future QIIME updates.

When defaults are loaded, all three locations are checked in order of precedence. Lowest precedence is given to the Qiime/qiime_config file, as these are defaults defined by the QIIME development team and are likely not relevant to other users' environments. Higher precedence is given to the file specified by $QIIME_CONFIG_FP, and this is envisioned to be used for defining system-wide defaults. Finally, highest precedence is given to ~/.qiime_config, so users have the ability to overwrite defaults defined elsewhere to have maximum control over their environment (e.g., if testing an experimental version of their 'cluster_jobs' script). Note that these values are defaults: the scripts typically allow overwriting of these values via their command line interfaces.

Note that users can have up to three separate qiime_config files, and one is provided by default with QIIME. At least one qiime_config file must be present in one of the three locations, or scripts that rely on qiime_config file will raise an error. Not all values need to be defined in all qiime_config files, but all values must be defined at least once. This is one more reason why you should not edit or remove Qiime/qiime_config: when new values are added in the future they will be defined in Qiime's default copy, but not in your local copies.

There is a script that prints the current qiime config settings in the scripts folder:

```
$ python Qiime/scripts/print_qiime_config.py
```

4.3 Denoising of 454 Data Sets

QIIME script: ***pyronoise.py***

The pyrosequencing technology employed by 454 sequencing machines produce characteristic sequencing errors, mostly imprecise signals for longer homopolymers runs. Most of the sequences contain no or only a few errors, but a few sequences contain enough errors to be classified as an additional rare OTU. The goal for the denoising procedure is to reduce the amount of erroneous OTUs and thus increasing the accuracy of the whole QIIME pipeline.

Currently, QIIME supports denoising using Chris Quince's PyroNoise (Quince et al., 2009), which needs to be installed separately.

The input to the denoising script is a textual representation of 454's .sff files, produced by 454's own tool sffinfo from the initial .sff file:

```
$ sffinfo 454Reads.sff > 454Reads.sff.txt
```

Input Arguments:

-i SFF_FP, --input_file=SFF_FP [REQUIRED]

- This is the path to the flowgram file (*.sff.txt).

-o OUTPUT_DIR, --output_dir=OUTPUT_DIR [Default: pyronoise_picked_otus/]

- This is the location where the resulting output should be written.

-n NUM_CPUS, --num_cpus=NUM_CPUS [Default: 1]

- This is the number of CPUs that should be used.

-s PRECISION, --precision=PRECISION [Default: 15.0]

- This is the precision that should be used (passed to pyroNoise).

-c CUT_OFF, --cut-off=CUT_OFF [Default: 0.05]

- This is the cut-off that should be used (passed to pyroNoise).

-k, --keep_intermediates [Default: False]

- If this parameter is passed, then the script will not delete intermediate PyroNoise files - which is useful for debugging.

Output:

The output of this script produces two files 1) a denoised FASTA set of cluster centroids and 2) an OTU mapping of flowgram identifiers to centroids.

Examples:

To denoise the flowgram sequences in 454Reads.sff.txt, you can use the following command:

```
$ python $qdir/pyronoise.py -i 454Reads.sff.txt -o 454Reads_out/
```

which produces these two output files:

- 454Reads.fasta: A denoised set of cluster centroids.
- 454Reads_otu.txt: A mapping of flowgram identifiers to centroids

On a multi-processor machine pyroNoise can be run in parallel using mpirun, where the number of processors is passed to the script via -n, as shown by the following comm:

```
$ python $qdir/pyronoise.py -i 454Reads.sff.txt -o 454Reads_out/ -n 4
```

Since PyroNoise's steep computational requirement, you should limit the application to small data sets. Barcodes and primers are not taken into account here, and barcoded samples should be denoised in separate steps. See Chris's PyroNoise web site for details or use a combination of ***split_libraries.py*** and sfffile (from the 454 software package) to separate the sequences into different sets.

References:

Altschul, S. F., Gish, W., Miller, W., Myers, E. W., & Lipman, D. J. (1990). Basic local alignment search tool. *J Mol Biol, 215*(3), 403-410.

Caporaso, J. G., Bittinger, K., Bushman, F. D., Desantis, T. Z., Andersen, G. L., & Knight, R. (2009). PyNAST: A flexible tool for aligning sequences to a template alignment. *Bioinformatics*.

Chen, V. B., Davis, I. W., & Richardson, D. C. (2009). KING (Kinemage, Next Generation): a versatile interactive molecular and scientific visualization program. *Protein Sci, 18*(11), 2403-2409.

DeSantis, T. Z., Hugenholtz, P., Larsen, N., Rojas, M., Brodie, E. L., Keller, K., et al. (2006). Greengenes, a chimera-checked 16S rRNA gene database and workbench compatible with ARB. *Appl Environ Microbiol, 72*(7), 5069-5072.

Edgar, R. C. (2004). MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Res, 32*(5), 1792-1797.

Hamady, M., & Knight, R. (2009). Microbial community profiling for human microbiome projects: Tools, techniques, and challenges. *Genome Res, 19*(7), 1141-1152.

Li, W., & Godzik, A. (2006). Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences. *Bioinformatics, 22*(13), 1658-1659.

Li, W., Jaroszewski, L., & Godzik, A. (2001). Clustering of highly homologous sequences to reduce the size of large protein databases. *Bioinformatics, 17*(3), 282-283.

Nawrocki, E. P., Kolbe, D. L., & Eddy, S. R. (2009). Infernal 1.0: inference of RNA alignments. *Bioinformatics, 25*(10), 1335-1337.

Parameswaran, P., Jalili, R., Tao, L., Shokralla, S., Gharizadeh, B., Ronaghi, M., et al. (2007). A pyrosequencing-tailored nucleotide barcode design unveils opportunities for large-scale sample multiplexing. *Nucleic Acids Res, 35*(19), e130.

Price, M. N., Dehal, P. S., & Arkin, A. P. (2009). FastTree: computing large minimum evolution trees with profiles instead of a distance matrix. *Mol Biol Evol, 26*(7), 1641-1650.

Quince, C., Lanzen, A., Curtis, T. P., Davenport, R. J., Hall, N., Head, I. M., et al. (2009). Accurate determination of microbial diversity from 454 pyrosequencing data. *Nat Methods, 6*(9), 639-641.

Reeder, J., & Knight, R. (2009). The 'rare biosphere': a reality check. *Nat Methods, 6*(9), 636-637.

Schloss, P. D., Westcott, S. L., Ryabin, T., Hall, J. R., Hartmann, M., Hollister, E. B., et al. (2009). Introducing mothur: open-source, platform-independent, community-supported software for describing and comparing microbial communities. *Appl Environ Microbiol, 75*(23), 7537-7541.

Shannon, P., Markiel, A., Ozier, O., Baliga, N. S., Wang, J. T., Ramage, D., et al. (2003). Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome Res, 13*(11), 2498-2504.

Sheneman, L., Evans, J., & Foster, J. A. (2006). Clearcut: a fast implementation of relaxed neighbor joining. *Bioinformatics, 22*(22), 2823-2824.

Wang, Q., Garrity, G. M., Tiedje, J. M., & Cole, J. R. (2007). Naive Bayesian classifier for rapid assignment of rRNA sequences into the new bacterial taxonomy. *Appl Environ Microbiol, 73*(16), 5261-5267.