Documentation for my collection of R functions. Mainly for me to remember, but may open up to others one day.

All functions written in R v3.6.2.

Other required packages and versions used for building:

ggplot2 v3.3.0

reshape2 v1.4.3

DECIPHER v2.14.0

stringr v1.4.0

pr2database v4.12.0

dada2 v.1.12.1

Citations:

**Data pre-processing functions**

*LCA2df*

**Requires:**

DECIPHER R package

**Use:**

*x 🡨 LCA2df(lcaer, rubric)*

**Overview:**

Converts .csv files output by MEGAN’s LCA algorithm into a dataframe formatted for use with my taxonomy pipeline.

**Inputs:**

1. *lcaer* = the R object (a dataframe) generated by the following command: *read.csv(file = “[your MEGAN output file name]”, header = FALSE, stringsAsFactors = FALSE)*.
2. *rubric* = the R object (a DNAStringSet; see package DECIPHER) generated by the following command: *readDNAStringSet(“[your query Fasta file of ASV sequences used for BLAST search]”)*.

**Outputs:**

1. a dataframe (*nrow* = number of ASVs, *ncol* = number of taxonomic ranks with assignments output by MEGAN LCA + 2) containing the names and sequences of your ASVs (names come from headers of your rubric fasta file), and corresponding taxonomic assignments by LCA.

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

*idtax2df*

**Requires:**

**Use:**

*x 🡨 idtax2df(taxon.object, boot)*

**Overview:**

Converts “taxon” objects output by the implementation of the idtaxa algorithm in the DADA2 pipeline (see: ).

**Inputs:**

1. *taxon.object* = the taxon object output by implementation of the idtaxa algorithm (see: ).
2. *boot* = the confidence threshold below which you’d like to NA-out assignments. Set to 0 if you’d like to forego.
3. *return.conf* = either “yes” or “no”, if “yes”, return a second dataframe with corresponding confidence estimates for each assignment in taxonomy dataframe.

**Outputs:**

1. if *return.conf = “no”*: a dataframe (*nrow* = number of ASVs, *ncol* = number of taxonomic ranks with assignments output by idtaxa).
2. if *return.conf = “yes”*: a list with element *[[1]]* containing the dataframe described above (see output 1), and element *[[2]]* containing a second dataframe with corresponding confidence estimates for each taxonomic assignment in *[[1]]*

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

*pr2\_tax\_miner*

**Requires:**

pr2database R package

**Use:**

*pr2\_tax\_miner(out.file)*

**Overview:**

Mines all unique taxonomic paths from the most recent update of the pr2database R package (built with pr2 v.4.12.0) and saves to a .csv file for use in taxonomic nomenclature mapping.

**Inputs:**

1. *out.file* = a string of the complete file name for your output .csv file.

**Outputs:**

1. saves a .csv file to disk according to inputs

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

**Taxonomy and trait mapping functions**

*taxmapper*

**Requires:**

Valid synonyms file path, valid outfilez paths.

**Use:**

*x 🡨 taxmapper(taxin, tax2map2, exceptions=c(“Bacteria”, “Archaea”), ignore.format=TRUE,*

*synonym.file=”tax\_synonyms\_FINAL.csv”, outfilez=”none”)*

**Overview:**

Maps a taxonomy table onto another taxonomy table by exact name-matching, regardless of rank.

**Inputs:**

1. *taxin* = a dataframe containing ASV sequences and a set of taxonomic assignments
2. *tax2map2* = dataframe of taxonomic nomenclature you’d like to “translate” taxin into
3. *exceptions* = character vector of taxonomy names for the function to skip searching for as they’re guaranteed to not be found in tax2map2
4. *ignore.format* = Boolean input to preprocess the taxonomy name removing underscores, hyphens, and uppercase before searching in tax2map2 (TRUE)
5. *synonym.file* = filename containing synonyms to check if exact name matching doesn’t work
6. *outfilez* = character vector specifying names of output csv files. if none, it won’t write any outputs

**Outputs:**

1. dataframe of unique taxonomic assignments from taxin mapped to tax2map2
2. character vector of taxonomy names from taxin that didn’t find a corresponding match to tax2map2
3. dataframe of taxin including its ASV’s and svN’s with its corresponding mapped taxonomic assignments from tax2map2

**Mapping Description:**

The algorithm maps one taxonomy table’s taxonomic nomenclature onto another by exact name matching, regardless of the ranking convention employed. The exactness of name matching can be relaxed by incorporating known taxonomic synonyms for particular names, and by inputting a flag in the function call to ignore formatting of taxonomic names. If specified, the option to ignore formatting removes sensitivity to case and other special characters including underscores and hyphens. It iterates through each ASV starting from its most specific ranking column toward its most generic ranking column to find its matching row from tax2map2 by exact name matching. For a specific ASV that has no matching after iterating through all of its taxonomy names, it doesn’t get added into the mapped ASV taxonomy table from the output.

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

*traitmapper\_Ramond*

**Requires:**

**Use:**

*x 🡨 traitmapper\_Ramond(taxin, map2,*

*map2.taxnames = c("Lineage1","Lineage2","Lineage3","Lineage4",*

*"Lineage5","Lineage6","Lineage7","Lineage8",*

*"Lineage9","Lineage10","Lineage11","Lineage12",*

*"Fam","Taxogroup","Taxo1","Last"),*

*dont.map = c("Eukaryota", "Archaea", "Bacteria",*

*"Alveolata", "Opisthokonta", "Archaeplastida", "Excavata", "Rhizaria", "Stramenopiles", "Hacrobia", "Amoebozoa", "Apusozoa", "Eukaryota\_X", "Protalveolata", "Terrabacteria"),*

*filezout = "none")*

**Overview:**

Maps a taxonomy table onto a trait database. See details below for mapping behavior.

**Inputs:**

1. *taxin* = a dataframe, the taxonomy table you would like to map to traits. Factors not supported.
2. *map2* = a dataframe, the trait database you’d like to map *taxin* onto. Factors not supported.
3. *map2.taxnames* = a character vector of column names of *map2* that contain taxonomic annotations to map the names in *taxin* onto.
4. *dont.map* = a character vector of taxonomic names within *taxin* that don’t require mapping. Default is all unique “kingdom” and “supergroup” assignments in the pr2 database (v.4.12.0). Include names of higher-order groups that will have highly ambiguous mapping outcomes; if these sorts of names are not included here, the computation speed will be dramatically reduced (it may never finish) and memory requirements will dramatically increase.
5. *filezout* = a character vector with, in order, the names of the output .csv files of the mapping result and the taxonomic names that were not able to be mapped. You can suppress writing these csv files to disk by setting *filezout = “none”*

**Outputs:**

1. a 2-element list with element *[[1]]* containing a dataframe (*nrow = nrow(taxin)*, *ncol = ncol(taxin) + ncol(map2)* + 2) of the mapping results, and [[2]] containing a character vector of the taxonomic names of taxin that were not able to be mapped to the columns *map2.taxnames* in *map2*
2. if *filezout != “none”*, 2 csv files (corresponding to the 2 elements of output (1) above) are written to disk

**Mapping Description:**

The algorithm maps a collection of taxonomic assignments onto a trait database (a collection of taxonomies annotated with functional traits). It does this for each ASV by iteratively searching, starting with the name in the ASV’s most resolved taxonomic rank (assumed to be at *taxin[,ncol(taxin)]*), for identical names, regardless of rank, in the supplied trait database. If an identical name is found in the trait database, the corresponding trait assignments are assigned to the ASV. If more than one match is found in the trait database, all unique trait assignments for each trait are compiled into a single string, and assigned collectively to the ASV in question. If the name is not found in the trait database, the algorithm proceeds to the next most highly-resolved taxonomic rank and repeats the search, assigning compiled trait annotations to each ASV based on the most highly-resolved taxonomic name found in the trait database. Taxonomic names that are unable to be mapped are stored in a separate array and returned for further analysis. If an ASV has no taxonomic names that are able to be mapped, all trait assignments remain unidentified.

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

*analyze\_traitmap\_byTrait*

**Requires:**

ggplot2, reshape2, and stringr R packages

Output of *traitmapper\_Ramond*

**Use:**

*x 🡨 analyze\_traitmap\_byTrait(map.result, trait.name, otu.table = "none", plotfilez = "none")*

**Overview:**

Analyzes trait mapping results produced by *traitmapper\_Ramond* for a specified trait to determine the proportion of ASVs and relative sequence abundances per sample that can be assigned to one or more particular trait categories, and that are assigned to each unique trait category. Produces up to 3 ggplot objects and 3 dataframes summarizing trait assignment results depending on input arguments.

**Inputs:**

1. *map.result* = a dataframe containing mapping results output by *traitmapper\_Ramond*
2. *trait.name* = a string indicating the trait (column name of map.result) to analyze
3. *otu.table* = a classic OTU table as generated by *removeBimeraDenovo* from the *dada2* package. Columns should be individual OTU/ASVs, rows should be samples. *colnames(otu.table)* should be identical to *map.result$ASV*. Sequence read counts can be supplied with or without normalization, but relative abundances are computed is any value in the matrix is > 1. This is used to analyze the proportion of sequence reads that can or cannot be assigned to a particular trait. If you prefer to omit this analysis, set *otu.table = “none”*
4. *plotfilez* = a character vector specifying the name(s) of pdf files to save the ggplot object. If otu.table is not provided, must be a *length = 1* character vector, or “none” to forego saving files. If otu.table is provided, must be a *length = 3* character vector, or “none” to forego saving files.

**Outputs:**

1. a 6-element list containing
   * *[[1]]* = a ggplot barplot summarizing the proportion of total ASVs assigned to each unique trait category (or combination of categories for ASVs that were mapped to more than one hit in the trait database)
   * *[[2]]* = a ggplot boxplot summarizing the distribution of relative sequence read abundances each trait category in each sample. *NULL* if otu.table is not supplied.
   * *[[3]]* = a ggplot histogram summarizing the distribution of cumulative relative sequence read abundances that were unable to be assigned to a single trait category in each sample. *NULL* if otu.table is not supplied.
   * *[[4]]* = a dataframe of row (column) indices of in *map.result* (*otu.table*) of all ASVs in each trait category
   * *[[5]]* = the dataframe used to create the plot in element *[[1]]* of the output list
   * *[[6]]* = a dataframe containing the cumulative relative sequence read abundances of each unique trait category (columns) in each sample (rows)
2. Up to 3 pdf files containing ggplot graphics written to disk

**Taxonomy table comparison functions**

*compare\_taxrez*

Requires:

Use:

Overview:

Inputs:

Outputs:

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

*compare\_byRank\_2way*

Requires:

Use:

Overview:

Inputs:

Outputs:

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

*compare\_byRank\_3way*

Requires:

Use:

Overview:

Inputs:

Outputs:

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

*compare\_assignments\_2way*

Requires:

Use:

Overview:

Inputs:

Outputs:

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

*compare\_assignments\_3way*

Requires:

Use:

Overview:

Inputs:

Outputs:

**Ensemble taxonomic assignment functions**

*consensus\_tax\_bestRez*

Requires:

Use:

Overview:

This function merges multiple taxonomy arrays into a single taxonomy array incorporating information from all input taxonomy arrays. For each ASV, the most highly resolved taxonomic assignment amongst the input taxonomy tables is assigned as the ensemble taxonomic assignment. In the event of a tie where two or more taxonomy tables have equivalent resolutions in their assignments, the tied taxonomy tables are compared. If identical assignments are found amongst them, this assignment is retained in the ensemble assignment. If there are any disagreements, a series of user-specified rules is followed to break the ties (see below for options).

Inputs:

1. *…* = an arbitrary number of taxonomy tables (dataframes). They should be identical dimensions with OTUs/ASVs as rows and taxonomic ranks as columns. The order of ASVs should be the same across all tables. *NA* should be used to indicate “not assigned”.
2. *tablenames* = a character vector including the names of each taxonomy dataframe supplied in *…*
3. *ranknamez* = the names of each rank (column) of the input dataframes. The output consensus taxonomy will have these as column names.
4. *tiebreakz* = a 1x1 character vector or a list of 1x2 character vectors. If a 1x1 character vector, it must be either “none” or “LCAlike”. If a list of 1x2 character vectors, *tiebreakz*[[X]][1] should match one the entries in tablenames, and *tiebreakz*[[X]][2] should be NA or should match any taxonomic name in *tiebreakz*[[X]][1]. These values indicate the taxonomy table to prioritize for all remaining tiebreakers (if *tiebreakz*[[X]][2] is NA) or for a particular taxonomic group (if *tiebreakz*[[X]][2] is a taxonomic name). The list should be in order of the priority in which you want the rules applied. By supplying a series of taxonomy table names alongside taxonomic names, you can thus prioritize a series of names before prioritizing a single taxonomy table for the remaining names.

Outputs:

A list including the following:

* [[1]] = dataframe containing the consensus taxonomy table
* [[2]] = a list with each of the original taxonomy dataframes
* [[3]] = a vector containing the row indices of the taxonomy tables that require further tie-breaking. If there were none given the input rules, this is an empty array.

Taxonomy merging rules:

1. For each ASV, the most resolved taxonomy (lowest number of NA ranks) is prescribed to the output.
2. If multiple input taxonomy arrays have identical resolution and taxonomic name assignments, the common taxonomic path is simply prescribed to the output.
3. If ASVs (rows) have equivalent resolution with non-equivalent names across multiple taxonomy arrays, a series of tie-breakers are used to determine the final taxonomic assignment for that ASV. Tiebreakers are user-specified and can include the following:
   1. “none” – no tie-breaking is done, and the output taxonomy array will be entirely unassigned at points where taxonomy tables have equal resolution but non-equal names at any rank
   2. “LCAlike” – a search is conducted to determine the rank (if any) at which the taxonomy arrays with equivalent resolution agree. If one is found, the ranks with names in agreement are used in the final output array, and all further ranks are left unassigned.
      1. “LCAlike” can be included at any position in the list in 3c below and will be prioritized relative to other entries in the list according to it’s position in the list
   3. A list containing specific instructions for prioritizing particular taxonomy arrays (and taxonomic groups within them) to break ties. See input arguments for details.

*consensus\_tax\_mostCom*

**Requires:**

All taxonomy tables inputted to have the same number of rows and columns and order of ASVs.

Rank names inputted matches the ranking column names in each inputted taxonomy table.

Length of weights vector to be the same number of taxonomy tables inputted.

Use:

**Overview:**

This function merges multiple taxonomy arrays into a single taxonomy array incorporating information from all input taxonomy arrays. This function merges multiple taxonomy arrays into a single taxonomy array incorporating information from all input taxonomy arrays. For each ASV, the most common taxonomy name amongst the inputted taxonomy tables at each rank is assigned as the ensemble taxonomic name. In the event of a tie where two or more taxonomy tables have equivalent frequencies, the tiebreakers inputted by the user is used to resolve it. If it can’t be resolved, an NA will be placed.

**Inputs:**

1. *…* = an arbitrary number of taxonomy tables (dataframes). They should be identical dimensions with OTUs/ASVs as rows and taxonomic ranks as columns. The order of ASVs should be the same across all tables. *NA* should be used to indicate “not assigned”.
2. *tablenames* = a character vector including the names of each taxonomy dataframe supplied in *…*
3. *ranknamez* = the names of each rank (column) of the input dataframes. The output consensus taxonomy will have these as column names.
4. *tiebreakz* = a list of character vectors with length of two in the format of c(“tablename”, “taxonomy name or NA”) in the order of priority. The highest priority will be the first element of the list while the lowest priority is the last element of the list. NA will be referred to prioritizing the specified table name as a whole while ‘na’ will be referred to prioritizing the taxonomy name “NA”.
5. *count.na =* a boolean input to have NA considered as a candidate in determining the majority taxonomy name
6. *trueMajority* = a boolean input to determine whether to relax the rule that a true majority has to occur at least 50% out of the pool of taxonomy names considered for ensemble (FALSE) or not (TRUE)
7. *weights* = a numeric vector assignment the “weight” of each taxonomy name from each taxonomy table. “weight” is defined as the number of times a taxonomy name from its corresponding table to repeat when considered to be the majority

**Outputs:**

dataframe containing the consensus taxonomy table

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~