Package 'metacoder'

January 6, 2018

Title Tools for Parsing, Manipulating, and Graphing Taxonomic Abundance Data

Version 0.2.0

Maintainer Zachary Foster <zacharyfoster1989@gmail.com>

Description A set of tools for parsing, manipulating, and graphing data classified by a hierarchy (e.g. a taxonomy).

Depends R (>= 3.0.2), taxa

License GPL-2 | GPL-3

LazyData true

URL https://grunwaldlab.github.io/metacoder_documentation/

BugReports https://github.com/grunwaldlab/metacoder/issues

Imports stringr, ggplot2, igraph, scales, grid, taxize, seqinr, reshape2, zoo, traits, RColorBrewer, RCurl, ape, reshape, stats, grDevices, utils, lazyeval, dplyr, magrittr, readr, rlang, biomformat, phylotate, ggfittext, vegan, ggrepel, cowplot, GA, Rcpp

Suggests knitr, rmarkdown, testthat, BiocInstaller, zlibbioc

VignetteBuilder knitr

RoxygenNote 6.0.1

Date 2018-01-04

Encoding UTF-8

biocViews

LinkingTo Rcpp

NeedsCompilation yes

Author Zachary Foster [aut, cre], Niklaus Grunwald [ths]

Repository CRAN

Date/Publication 2018-01-05 23:53:47 UTC

R topics documented:

`calc_n_samples` *Count the number of samples*

Description

For a given table in a `taxmap` object, count the number of samples with greater than zero occurrences.

Usage

```
calc_n_samples(obj, dataset, cols = NULL, groups = NULL, out_names = NULL,
  drop = FALSE, append = FALSE)
```

Arguments

`obj`	A `taxmap` object
`dataset`	The name of a table in `obj` that contains counts.
`cols`	The names/indexes of columns in `data` that have counts. By Default, all numeric columns in `data` are used.
`groups`	Group counts of multiple columns per treatment/group. This should be a vector of group IDs (e.g. character, integer) the same length as `cols` that defines which samples go in which group. When used, there will be one column in the output for each unique value in `groups`.
`out_names`	The names of count columns in the output. Must be the length 1 or same length as `unique(groups)`, if used.
`drop`	If `groups` is not used, return a vector of the results instead of a table with one column.
`append`	If `TRUE`, append results to input table and return.

Value

A tibble

See Also

Other calculations: `calc_obs_props`, `calc_taxon_abund`, `compare_groups`, `rarefy_obs`, `zero_low_counts`

Examples

```
## Not run:
# Parse dataset for example
x = parse_tax_data(hmp_otus, class_cols = "lineage", class_sep = ";",
                   class_key = c(tax_rank = "info", tax_name = "taxon_name"),
                   class_regex = "^(.+)__(.+)$")

# Count samples with reads
```

```
calc_n_samples(x, dataset = "tax_data")

# Return a vector instead of a table
calc_n_samples(x, dataset = "tax_data", drop = TRUE)

# Only use some columns
calc_n_samples(x, dataset = "tax_data", cols = hmp_samples$sample_id[1:5])

# Return a count for each treatment
calc_n_samples(x, dataset = "tax_data", groups = hmp_samples$body_site)

# Rename output columns
calc_n_samples(x, dataset = "tax_data", groups = hmp_samples$body_site,
               out_names = c("A", "B", "C", "D", "E"))

# Add results to input table
calc_n_samples(x, dataset = "tax_data", append = TRUE)

## End(Not run)
```

`calc_obs_props` *Calculate proportions from observation counts*

Description

For a given table in a `taxmap` object, convert one or more columns containing counts to proportions. This is meant to be used with counts associated with observations (e.g. OTUs), as opposed to counts that have already been summed per taxon.

Usage

```
calc_obs_props(obj, dataset, cols = NULL, other_cols = FALSE,
  out_names = NULL)
```

Arguments

`obj`	A `taxmap` object
`dataset`	The name of a table in `obj` that contains counts.
`cols`	The names/indexes of columns in `dataset` to use. By default, all numeric columns are used. Takes one of the following inputs: **TRUE/FALSE:** All/No columns will used. **Character vector:** The names of columns to use **Numeric vector:** The indexes of columns to use **Vector of TRUE/FALSE of length equal to the number of columns:** Use the columns corresponding to `TRUE` values.

other_cols — Preserve in the output non-target columns present in the input data. New columns with proportions will always be on the end. The "taxon_id" column will always be preserved in the front. Takes one of the following inputs:

TRUE/FALSE: All non-target columns will be preserved or not.

Character vector: The names of columns to preserve

Numeric vector: The indexes of columns to preserve

Vector of TRUE/FALSE of length equal to the number of columns: Preserve the columns corresponding to TRUE values.

out_names — If supplied, rename the output proportion columns. Must be the same length as cold.

Value

A tibble

See Also

Other calculations: calc_n_samples, calc_taxon_abund, compare_groups, rarefy_obs, zero_low_counts

Examples

```
## Not run:
# Parse dataset for examples
x = parse_tax_data(hmp_otus, class_cols = "lineage", class_sep = ";",
                   class_key = c(tax_rank = "info", tax_name = "taxon_name"),
                   class_regex = "^(.+)__(.+)$")

# Calculate proportions for all numeric columns
calc_obs_props(x, "tax_data")

# Calculate proportions for a subset of columns
calc_obs_props(x, "tax_data", cols = c("700035949", "700097855", "700100489"))
calc_obs_props(x, "tax_data", cols = 4:6)
calc_obs_props(x, "tax_data", cols = startsWith(colnames(x$data$tax_data), "70001"))

# Including all other columns in ouput
calc_obs_props(x, "tax_data", other_cols = TRUE)

# Inlcuding specific columns in output
calc_obs_props(x, "tax_data", cols = c("700035949", "700097855", "700100489"),
               other_cols = 2:3)

# Rename output columns
calc_obs_props(x, "tax_data", cols = c("700035949", "700097855", "700100489"),
               out_names = c("a", "b", "c"))


## End(Not run)
```

`calc_taxon_abund` *Sum observation values for each taxon*

Description

For a given table in a `taxmap` object, sum the values in each column for each taxon. This is useful to convert per-observation counts (e.g. OTU counts) to per-taxon counts.

Usage

```
calc_taxon_abund(obj, dataset, cols = NULL, groups = NULL,
  out_names = NULL)
```

Arguments

`obj`	A `taxmap` object
`dataset`	The name of a table in `obj` that contains counts.
`cols`	The names/indexes of columns in `dataset` to use. By default, all numeric columns are used. Takes one of the following inputs: **TRUE/FALSE:** All/No columns will used. **Character vector:** The names of columns to use **Numeric vector:** The indexes of columns to use **Vector of TRUE/FALSE of length equal to the number of columns:** Use the columns corresponding to `TRUE` values.
`groups`	Group counts of multiple columns per treatment/group. This should be a vector of group IDs (e.g. character, integer) the same length as `cols` that defines which samples go in which group. When used, there will be one column in the output for each unique value in `groups`.
`out_names`	The names of count columns in the output. Must be the same length as `cols` (or `unique(groups)`, if used).

Value

A tibble

See Also

Other calculations: `calc_n_samples`, `calc_obs_props`, `compare_groups`, `rarefy_obs`, `zero_low_counts`

Examples

```
## Not run:
# Parse dataset for example
x = parse_tax_data(hmp_otus, class_cols = "lineage", class_sep = ";",
                   class_key = c(tax_rank = "info", tax_name = "taxon_name"),
                   class_regex = "^(.+)__(.+)$")
```

```
# Calculate the taxon abundance for each numeric column (i.e. sample)
calc_taxon_abund(x, "tax_data")

# Calculate the taxon abundance for a subset of columns
calc_taxon_abund(x, "tax_data", cols = 4:5)
calc_taxon_abund(x, "tax_data", cols = c("700035949", "700097855"))
calc_taxon_abund(x, "tax_data", cols = startsWith(colnames(x$data$tax_data), "70001"))

# Calculate the taxon abundance for groups of columns (e.g. treatments)
#  Note that we do not need to use the "cols" option for this since all
#  numeric columns are samples in this dataset. If there were numeric columns
#  that were not samples present in hmp_samples, the "cols" would be needed.
calc_taxon_abund(x, "tax_data", groups = hmp_samples$sex)
calc_taxon_abund(x, "tax_data", groups = hmp_samples$body_site)

# The above example using the "cols" option, even though not needed in this case
calc_taxon_abund(x, "tax_data", cols = hmp_samples$sample_id,
                 groups = hmp_samples$sex)

# Rename the output columns
calc_taxon_abund(x, "tax_data", cols = hmp_samples$sample_id[1:10],
                 out_names = letters[1:10])
calc_taxon_abund(x, "tax_data", groups = hmp_samples$sex,
                 out_names = c("Women", "Men"))

# Geting a total for all columns
calc_taxon_abund(x, "tax_data", cols = hmp_samples$sample_id,
                 groups = rep("total", nrow(hmp_samples)))

## End(Not run)
```

`compare_groups` *Compare groups of samples*

Description

Apply a function to compare data, usually abundance, from pairs of treatments/groups. By default, every pairwise combination of treatments are compared. A custom function can be supplied to perform the comparison. The plotting function `heat_tree_matrix` is useful for visualizing these results.

Usage

```
compare_groups(obj, dataset, cols, groups, func = NULL, combinations = NULL,
  other_cols = FALSE)
```

Arguments

obj	A taxmap object
dataset	The name of a table in obj that contains data for each sample in columns.
cols	The names/indexes of columns in dataset to use. By default, all numeric columns are used. Takes one of the following inputs: **TRUE/FALSE:** All/No columns will used. **Character vector:** The names of columns to use **Numeric vector:** The indexes of columns to use **Vector of TRUE/FALSE of length equal to the number of columns:** Use the columns corresponding to TRUE values.
groups	A vector defining how samples are grouped into "treatments". Must be the same order and length as cols.
func	The function to apply for each comparison. For each row in dataset, for each combination of groups, this function will receive the data for each treatment, passed as two character vectors. Therefore the function must take at least 2 arguments corresponding to the two groups compared. The function should return a vector or list of results of a fixed length. If named, the names will be used in the output. The names should be consistent as well. A simple example is function(x, y) mean(x) - mean(y). By default, the following function is used:
combinations	Which combinations of groups to use. Must be a list of vectors, each containing the names of 2 groups to compare. By default, all pairwise combinations of groups are compared.
other_cols	If TRUE, preserve all columns not in cols in the output. If FALSE, dont keep other columns. If a column names or indexes are supplied, only preserve those columns.

```
function(abund_1, abund_2) {
  log_ratio <- log2(median(abund_1) / median(abund_2))
  if (is.nan(log_ratio)) {
    log_ratio <- 0
  }
  list(log2_median_ratio = log_ratio,
       median_diff = median(abund_1) - median(abund_2),
       mean_diff = mean(abund_1) - mean(abund_2),
       wilcox_p_value = wilcox.test(abund_1, abund_2)$p.value)
}
```

Value

A tibble

See Also

Other calculations: calc_n_samples, calc_obs_props, calc_taxon_abund, rarefy_obs, zero_low_counts

Examples

```
## Not run:
# Parse dataset for plotting
x = parse_tax_data(hmp_otus, class_cols = "lineage", class_sep = ";",
                   class_key = c(tax_rank = "info", tax_name = "taxon_name"),
                   class_regex = "^(.+)__(.+)$")

# Convert counts to proportions
x$data$otu_table <- calc_obs_props(x, dataset = "tax_data", cols = hmp_samples$sample_id)

# Get per-taxon counts
x$data$tax_table <- calc_taxon_abund(x, dataset = "otu_table", cols = hmp_samples$sample_id)

# Calculate difference between groups
x$data$diff_table <- compare_groups(x, dataset = "tax_table",
                                    cols = hmp_samples$sample_id,
                                    groups = hmp_samples$body_site)

# Plot results (might take a few minutes)
heat_tree_matrix(x,
                 dataset = "diff_table",
                 node_size = n_obs,
                 node_label = taxon_names,
                 node_color = log2_median_ratio,
                 node_color_range = diverging_palette(),
                 node_color_trans = "linear",
                 node_color_interval = c(-3, 3),
                 edge_color_interval = c(-3, 3),
                 node_size_axis_label = "Number of OTUs",
                 node_color_axis_label = "Log2 ratio median proportions")


## End(Not run)
```

`complement` *Find complement of sequences*

Description

Find the complement of one or more sequences stored as a character vector. This is a wrapper for `comp` for character vectors instead of lists of character vectors with one value per letter. IUPAC ambiguity code are handled and the upper/lower case is preserved.

Usage

```
complement(seqs)
```

Arguments

seqs	A character vector with one element per sequence.

See Also

Other sequence transformations: rev_comp, reverse

Examples

```
complement(c("aagtgGGTGaa", "AAGTGGT"))
```

---

diverging_palette *The default diverging color palette*

Description

Returns the default color palette for diverging data

Usage

```
diverging_palette()
```

Value

character of hex color codes

Examples

```
diverging_palette()
```

---

filter_ambiguous_taxa *Filter ambiguous taxon names*

Description

Filter out taxa with ambiguous names, such as "unknown" or "uncultured". NOTE: some parameters of this function are passed to filter_taxa with the "invert" option set to TRUE. Works the same way as filter_taxa for the most part.

Usage

```
filter_ambiguous_taxa(obj, unknown = TRUE, uncultured = TRUE,
  name_regex = ".", ignore_case = TRUE, subtaxa = FALSE,
  drop_obs = TRUE, reassign_obs = TRUE, reassign_taxa = TRUE)
```

Arguments

obj	A taxmap object
unknown	If TRUE, Remove taxa with names the suggest they are placeholders for unknown taxa (e.g. "unknown ...").
uncultured	If TRUE, Remove taxa with names the suggest they are assigned to uncultured organisms (e.g. "uncultured ...").
name_regex	The regex code to match a valid character in a taxon name. For example, "[a-z]" would mean taxon names can only be lower case letters.
ignore_case	If TRUE, dont consider the case of the text when determining a match.
subtaxa	(logical or numeric of length 1) If TRUE, include subtaxa of taxa passing the filter. Positive numbers indicate the number of ranks below the target taxa to return. 0 is equivalent to FALSE. Negative numbers are equivalent to TRUE.
drop_obs	(logical) This option only applies to taxmap() objects. If FALSE, include observations even if the taxon they are assigned to is filtered out. Observations assigned to removed taxa will be assigned to NA. This option can be either simply TRUE/FALSE, meaning that all data sets will be treated the same, or a logical vector can be supplied with names corresponding one or more data sets in obj$data. For example, c(abundance = FALSE, stats = TRUE) would include observations whose taxon was filtered out in obj$data$abundance, but not in obj$data$stats. See the reassign_obs option below for further complications.
reassign_obs	(logical of length 1) This option only applies to taxmap() objects. If TRUE, observations assigned to removed taxa will be reassigned to the closest supertaxon that passed the filter. If there are no supertaxa of such an observation that passed the filter, they will be filtered out if drop_obs is TRUE. This option can be either simply TRUE/FALSE, meaning that all data sets will be treated the same, or a logical vector can be supplied with names corresponding one or more data sets in obj$data. For example, c(abundance = TRUE, stats = FALSE) would reassign observations in obj$data$abundance, but not in obj$data$stats.
reassign_taxa	(logical of length 1) If TRUE, subtaxa of removed taxa will be reassigned to the closest supertaxon that passed the filter. This is useful for removing intermediate levels of a taxonomy.

Details

If you encounter a taxon name that represents an ambiguous taxon that is not filtered out by this function, let us know and we will add it.

Value

A taxmap object

`heat_tree_matrix` *Plot a matrix of heat trees*

Description

Plot a matrix of heat trees for showing parwise comparisons. A larger, labelled tree serves as a key for the matrix of smaller unlabelled trees. The data for this function is typically created with `compare_groups`,

Usage

```
heat_tree_matrix(obj, dataset, label_small_trees = FALSE, key_size = 0.6,
  seed = 1, ...)
```

Arguments

`obj`	A `taxmap` object
`dataset`	The name of a table in `obj$data` that is the output of `compare_groups` or in the same format.
`label_small_trees`	If `TRUE` add labels to small trees as well as the key tree. Otherwise, only the key tree will be labeled.
`key_size`	The size of the key tree relative to the whole graph. For example, 0.5 means half the width/height of the graph.
`seed`	That random seed used to make the graphs.
`...`	Passed to `heat_tree`. Some options will be overwritten.

Examples

```
## Not run:
# Parse dataset for plotting
x <- parse_tax_data(hmp_otus, class_cols = "lineage", class_sep = ";",
                    class_key = c(tax_rank = "info", tax_name = "taxon_name"),
                    class_regex = "^(.+)__(.+)$")

# Convert counts to proportions
x$data$otu_table <- calc_obs_props(x, dataset = "tax_data", cols = hmp_samples$sample_id)

# Get per-taxon counts
x$data$tax_table <- calc_taxon_abund(x, dataset = "otu_table", cols = hmp_samples$sample_id)

# Calculate difference between treatments
x$data$diff_table <- compare_groups(x, dataset = "tax_table",
                                    cols = hmp_samples$sample_id,
                                    groups = hmp_samples$body_site)

# Plot results (might take a few minutes)
```

```
heat_tree_matrix(x,
                 dataset = "diff_table",
                 node_size = n_obs,
                 node_label = taxon_names,
                 node_color = log2_median_ratio,
                 node_color_range = diverging_palette(),
                 node_color_trans = "linear",
                 node_color_interval = c(-3, 3),
                 edge_color_interval = c(-3, 3),
                 node_size_axis_label = "Number of OTUs",
                 node_color_axis_label = "Log2 ratio median proportions")


## End(Not run)
```

`hmp_otus` *A HMP subset*

Description

A subset of the Human Microbiome Project abundance matrix produced by QIIME. It contains OTU ids, taxonomic lineages, and the read counts for 50 samples. See `hmp_samples` for the matching dataset of sample information.

Format

A 1,000 x 52 tibble.

Details

The 50 samples were randomly selected such that there were 10 in each of 5 treatments: "Saliva", "Throat", "Stool", "Right_Antecubital_fossa", "Anterior_nares". For each treatment, there were 5 samples from men and 5 from women.

Source

Subset from data available at https://www.hmpdacc.org/hmp/HMQCP/

See Also

Other hmp_data: `hmp_samples`

hmp_samples *Sample information for HMP subset*

Description

The sample information for a subset of the Human Microbiome Project data. It contains the sample ID, sex, and body site for each sample in the abundance matrix stored in hmp_otus. The "sample_id" column corresponds to the column names of hmp_otus.

Format

A 50 x 3 tibble.

Details

The 50 samples were randomly selected such that there were 10 in each of 5 treatments: "Saliva", "Throat", "Stool", "Right_Antecubital_fossa", "Anterior_nares". For each treatment, there were 5 samples from men and 5 from women. "Right_Antecubital_fossa" was renamed to "Skin" and "Anterior_nares" to "Nose".

Source

Subset from data available at https://www.hmpdacc.org/hmp/HMQCP/

See Also

Other hmp_data: hmp_otus

is_ambiguous *Find ambiguous taxon names*

Description

Find taxa with ambiguous names, such as "unknown" or "uncultured".

Usage

```
is_ambiguous(taxon_names, unknown = TRUE, uncultured = TRUE,
  name_regex = ".", ignore_case = TRUE)
```

Arguments

taxon_names	A taxmap object
unknown	If TRUE, Remove taxa with names the suggest they are placeholders for unknown taxa (e.g. "unknown ...").
uncultured	If TRUE, Remove taxa with names the suggest they are assigned to uncultured organisms (e.g. "uncultured ...").
name_regex	The regex code to match a valid character in a taxon name. For example, "[a-z]" would mean taxon names can only be lower case letters.
ignore_case	If TRUE, dont consider the case of the text when determining a match.

Details

If you encounter a taxon name that represents an ambiguous taxon that is not filtered out by this function, let us know and we will add it.

Value

TRUE/FALSE vector corresponding to taxon_names

---

layout_functions *Layout functions*

Description

Functions used to determine graph layout. Calling the function with no parameters returns available function names. Calling the function with only the name of a function returns that function. Supplying a name and a graph object to run the layout function on the graph.

Usage

```
layout_functions(name = NULL, graph = NULL, intitial_coords = NULL,
  effort = 1, ...)
```

Arguments

name	(character of length 1 OR NULL) name of algorithm. Leave NULL to see all options.
graph	(igraph) The graph to generate the layout for.
intitial_coords	(matrix) Initial node layout to base new layout off of.
effort	(numeric of length 1) The amount of effort to put into layouts. Typically determines the the number of iterations.
...	(other arguments) Passed to igraph layout function used.

Value

The name available functions, a layout functions, or a two-column matrix depending on how arguments are provided.

Examples

```
# List available function names:
layout_functions()

# Execute layout function on graph:
layout_functions("davidson-harel", igraph::make_ring(5))
```

`metacoder` *Metacoder*

Description

A package for planning and analysis of amplicon metagenomics research projects.

Details

The goal of the `metacoder` package is to provide a set of tools for:

- Standardized parsing of taxonomic information from diverse resources.
- Visualization of statistics distributed over taxonomic classifications.
- Evaluating potential metabarcoding primers for taxonomic specificity.
- Providing flexible functions for analyzing taxonomic and abundance data.

To accomplish these goals, `metacoder` leverages resources from other R packages, interfaces with external programs, and provides novel functions where needed to allow for entire analyses within R.

Documentation

The full documentation can be found online at http://grunwaldlab.github.io/metacoder_documentation.

There is also a short vignette included for offline use that can be accessed by the following code:

```
browseVignettes(package = "metacoder")
```

Plotting:

- `heat_tree`
- `heat_tree_matrix`

In silico PCR:

- primersearch

Analysis:

- calc_taxon_abund
- calc_obs_props
- rarefy_obs
- compare_groups
- zero_low_counts
- calc_n_samples
- filter_ambiguous_taxa

Parsers:

- parse_greengenes
- parse_mothur_tax_summary
- parse_mothur_taxonomy
- parse_newick
- parse_phyloseq
- parse_phylo
- parse_qiime_biom
- parse_rdp
- parse_silva_fasta
- parse_unite_general

Writers:

- write_greengenes
- write_mothur_taxonomy
- write_rdp
- write_silva_fasta
- write_unite_general

Database querying:

- ncbi_taxon_sample

Author(s)

Zachary Foster and Niklaus Grunwald

`ncbi_taxon_sample` *Download representative sequences for a taxon*

Description

Downloads a sample of sequences meant to evenly capture the diversity of a given taxon. Can be used to get a shallow sampling of vast groups. **CAUTION:** This function can make MANY queries to Genbank depending on arguments given and can take a very long time. Choose your arguments carefully to avoid long waits and needlessly stressing NCBI's servers. Use a downloaded database and a parser from the `taxa` package when possible.

Usage

```
ncbi_taxon_sample(name = NULL, id = NULL, target_rank, min_counts = NULL,
  max_counts = NULL, interpolate_min = TRUE, interpolate_max = TRUE,
  min_children = NULL, max_children = NULL, seqrange = "1:3000",
  getrelated = FALSE, fuzzy = TRUE, limit = 10, entrez_query = NULL,
  hypothetical = FALSE, verbose = TRUE)
```

Arguments

`name`	(`character` of length 1) The taxon to download a sample of sequences for.
`id`	(`character` of length 1) The taxon id to download a sample of sequences for.
`target_rank`	(`character` of length 1) The finest taxonomic rank at which to sample. The finest rank at which replication occurs. Must be a finer rank than `taxon`.
`min_counts`	(named `numeric`) The minimum number of sequences to download for each taxonomic rank. The names correspond to taxonomic ranks.
`max_counts`	(named `numeric`) The maximum number of sequences to download for each taxonomic rank. The names correspond to taxonomic ranks.
`interpolate_min`	(`logical`) If `TRUE`, values supplied to `min_counts` and `min_children` will be used to infer the values of intermediate ranks not specified. Linear interpolation between values of specified ranks will be used to determine values of unspecified ranks.
`interpolate_max`	(`logical`) If `TRUE`, values supplied to `max_counts` and `max_children` will be used to infer the values of intermediate ranks not specified. Linear interpolation between values of specified ranks will be used to determine values of unspecified ranks.
`min_children`	(named `numeric`) The minimum number sub-taxa of taxa for a given rank must have for its sequences to be searched. The names correspond to taxonomic ranks.
`max_children`	(named `numeric`) The maximum number sub-taxa of taxa for a given rank must have for its sequences to be searched. The names correspond to taxonomic ranks.

seqrange	(character) Sequence range, as e.g., "1:1000". This is the range of sequence lengths to search for. So "1:1000" means search for sequences from 1 to 1000 characters in length.
getrelated	(logical) If TRUE, gets the longest sequences of a species in the same genus as the one searched for. If FALSE, returns nothing if no match found.
fuzzy	(logical) Whether to do fuzzy taxonomic ID search or exact search. If TRUE, we use xXarbitraryXx[porgn:__txid<ID>], but if FALSE, we use txid<ID>. Default: FALSE
limit	(numeric) Number of sequences to search for and return. Max of 10,000. If you search for 6000 records, and only 5000 are found, you will of course only get 5000 back.
entrez_query	(character; length 1) An Entrez-format query to filter results with. This is useful to search for sequences with specific characteristics. The format is the same as the one used to seach genbank. (https://www.ncbi.nlm.nih.gov/books/NBK3837/#EntrezHelp.Entrez_Searching_Options)
hypothetical	(logical; length 1) If FALSE, an attempt will be made to not return hypothetical or predicted sequences judging from accession number prefixs (XM and XR). This can result in less than the limit being returned even if there are more sequences available, since this filtering is done after searching NCBI.
verbose	(logical) If TRUE, progress messages will be printed.

Examples

```
## Not run:

# Look up 5 ITS sequences from each fungal class
data <- ncbi_taxon_sample(name = "Fungi", target_rank = "class", limit = 5,
                          entrez_query = '"internal transcribed spacer"[All Fields]')

# Look up taxonomic information for sequences
obj <- lookup_tax_data(data, type = "seq_id", column = "gi_no")

# Plot information
filter_taxa(obj, taxon_names == "Fungi", subtaxa = TRUE) %>%
  heat_tree(node_label = taxon_names, node_color = n_obs, node_size = n_obs)

## End(Not run)
```

---

parse_greengenes *Parse Greengenes release*

Description

Parses the greengenes database.

Usage

```
parse_greengenes(tax_file, seq_file = NULL)
```

Arguments

tax_file	(character of length 1) The file path to the greengenes taxonomy file.
seq_file	(character of length 1) The file path to the greengenes sequence fasta file. This is optional.

Details

The taxonomy input file has a format like:

```
228054  k__Bacteria; p__Cyanobacteria; c__Synechococcophycideae; o__Synech...
844608  k__Bacteria; p__Cyanobacteria; c__Synechococcophycideae; o__Synech...
...
```

The optional sequence file has a format like:

```
>1111886
AACGAACGCTGGCGGCATGCCTAACACATGCAAGTCGAACGAGACCTTCGGGTCTAGTGGCGCACGGGTGCGTA...
>1111885
AGAGTTTGATCCTGGCTCAGAATGAACGCTGGCGGCGTGCCTAACACATGCAAGTCGTACGAGAAATCCCGAGC...
...
```

Value

taxmap

See Also

Other parsers: parse_mothur_tax_summary, parse_mothur_taxonomy, parse_newick, parse_phyloseq, parse_phylo, parse_qiime_biom, parse_rdp, parse_silva_fasta, parse_unite_general

---

parse_mothur_taxonomy *Parse mothur Classify.seqs *.taxonomy output*

Description

Parse the '*.taxonomy' file that is returned by the 'Classify.seqs' command in mothur. If confidence scores are present, they are included in the output.

Usage

```
parse_mothur_taxonomy(file = NULL, text = NULL)
```

Arguments

file	(character of length 1) The file path to the input file. Either "file" or "text" must be used, but not both.
text	(character) An alternate input to "file". The contents of the file as a character. Either "file" or "text" must be used, but not both.

Details

The input file has a format like:

```
AY457915 Bacteria(100);Firmicutes(99);Clostridiales(99);Johnsone...
AY457914 Bacteria(100);Firmicutes(100);Clostridiales(100);Johnso...
AY457913 Bacteria(100);Firmicutes(100);Clostridiales(100);Johnso...
AY457912 Bacteria(100);Firmicutes(99);Clostridiales(99);Johnsone...
AY457911 Bacteria(100);Firmicutes(99);Clostridiales(98);Ruminoco...
```

or...

```
AY457915 Bacteria;Firmicutes;Clostridiales;Johnsonella_et_rel.;J...
AY457914 Bacteria;Firmicutes;Clostridiales;Johnsonella_et_rel.;J...
AY457913 Bacteria;Firmicutes;Clostridiales;Johnsonella_et_rel.;J...
AY457912 Bacteria;Firmicutes;Clostridiales;Johnsonella_et_rel.;J...
AY457911 Bacteria;Firmicutes;Clostridiales;Ruminococcus_et_rel.;...
```

Value

taxmap

See Also

Other parsers: parse_greengenes, parse_mothur_tax_summary, parse_newick, parse_phyloseq, parse_phylo, parse_qiime_biom, parse_rdp, parse_silva_fasta, parse_unite_general

---

parse_mothur_tax_summary

*Parse mothur *.tax.summary Classify.seqs output*

Description

Parse the '*.tax.summary' file that is returned by the 'Classify.seqs' command in mothur.

Usage

```
parse_mothur_tax_summary(file = NULL, text = NULL, table = NULL)
```

Arguments

file	(character of length 1) The file path to the input file. Either "file", "text", or "table" must be used, but only one.
text	(character) An alternate input to "file". The contents of the file as a character. Either "file", "text", or "table" must be used, but only one.
table	(character of length 1) An already parsed data.frame or tibble. Either "file", "text", or "table" must be used, but only one.

Details

The input file has a format like:

```
taxlevel  rankID  taxon  daughterlevels  total A B C
0 0 Root 2 242 84 84 74
1 0.1 Bacteria 50 242 84 84 74
2 0.1.2 Actinobacteria 38 13 0 13 0
3 0.1.2.3 Actinomycetaceae-Bifidobacteriaceae 10 13 0 13 0
4 0.1.2.3.7 Bifidobacteriaceae 6 13 0 13 0
5 0.1.2.3.7.2 Bifidobacterium_choerinum_et_rel. 8 13 0 13 0
6 0.1.2.3.7.2.1 Bifidobacterium_angulatum_et_rel. 1 11 0 11 0
7 0.1.2.3.7.2.1.1 unclassified 1 11 0 11 0
8 0.1.2.3.7.2.1.1.1 unclassified 1 11 0 11 0
9 0.1.2.3.7.2.1.1.1.1 unclassified 1 11 0 11 0
10 0.1.2.3.7.2.1.1.1.1.1 unclassified 1 11 0 11 0
11 0.1.2.3.7.2.1.1.1.1.1.1 unclassified 1 11 0 11 0
12 0.1.2.3.7.2.1.1.1.1.1.1.1 unclassified 1 11 0 11 0
6 0.1.2.3.7.2.5 Bifidobacterium_longum_et_rel. 1 2 0 2 0
7 0.1.2.3.7.2.5.1 unclassified 1 2 0 2 0
8 0.1.2.3.7.2.5.1.1 unclassified 1 2 0 2 0
9 0.1.2.3.7.2.5.1.1.1 unclassified 1 2 0 2 0
```

or

```
taxon total A B C
"k__Bacteria";"p__Actinobacteria";"c__Actinobacteria";... 1 0 1 0
"k__Bacteria";"p__Actinobacteria";"c__Actinobacteria";... 1 0 1 0
"k__Bacteria";"p__Actinobacteria";"c__Actinobacteria";... 1 0 1 0
```

Value

taxmap

See Also

Other parsers: parse_greengenes, parse_mothur_taxonomy, parse_newick, parse_phyloseq, parse_phylo, parse_qiime_biom, parse_rdp, parse_silva_fasta, parse_unite_general

parse_newick *Parse a Newick file*

Description

Parse a Newick file into a taxmap object.

Usage

```
parse_newick(file)
```

Arguments

file	(character of length 1) The file path to the input file.

Details

The input file has a format like:

```
(ant:17, (bat:31, cow:22):7, dog:22, (elk:33, fox:12):40);
(dog:20, (elephant:30, horse:60):20):50;
```

Value

taxmap

See Also

Other parsers: parse_greengenes, parse_mothur_tax_summary, parse_mothur_taxonomy, parse_phyloseq, parse_phylo, parse_qiime_biom, parse_rdp, parse_silva_fasta, parse_unite_general

parse_phylo *Parse a phylo object*

Description

Parses a phylo object from the ape package.

Usage

```
parse_phylo(obj)
```

Arguments

obj	A phylo object from the ape package.

Value

taxmap

See Also

Other parsers: parse_greengenes, parse_mothur_tax_summary, parse_mothur_taxonomy, parse_newick, parse_phyloseq, parse_qiime_biom, parse_rdp, parse_silva_fasta, parse_unite_general

---

parse_phyloseq *Convert a phyloseq to taxmap*

Description

Converts a phyloseq object to a taxmap object.

Usage

```
parse_phyloseq(obj)
```

Arguments

obj	A phyloseq object

Value

A taxmap object

See Also

Other parsers: parse_greengenes, parse_mothur_tax_summary, parse_mothur_taxonomy, parse_newick, parse_phylo, parse_qiime_biom, parse_rdp, parse_silva_fasta, parse_unite_general

Examples

```
## Not run:

# Install phyloseq to get example data
# source('http://bioconductor.org/biocLite.R')
# biocLite('phyloseq')

# Parse example dataset
library(phyloseq)
data(GlobalPatterns)
x <- parse_phyloseq(GlobalPatterns)

# Plot data
heat_tree(x,
          node_size = n_obs,
          node_color = n_obs,
```

```
            node_label = taxon_names,
            tree_label = taxon_names)


## End(Not run)
```

`parse_qiime_biom` *Parse a BIOM output from QIIME*

Description

Parses a file in BIOM format from QIIME into a taxmap object. I have not tested if it works with other BIOM files.

Usage

```
parse_qiime_biom(file)
```

Arguments

`file`	(`character` of length 1) The file path to the input file.

Details

This function was inspired by the tutorial created by Geoffrey Zahn at http://geoffreyzahn.com/getting-your-otu-table-into-r/.

Value

A taxmap object

See Also

Other parsers: `parse_greengenes`, `parse_mothur_tax_summary`, `parse_mothur_taxonomy`, `parse_newick`, `parse_phyloseq`, `parse_phylo`, `parse_rdp`, `parse_silva_fasta`, `parse_unite_general`

parse_rdp *Parse RDP FASTA release*

Description

Parses an RDP reference FASTA file.

Usage

```
parse_rdp(input = NULL, file = NULL, include_seqs = TRUE,
  add_species = FALSE)
```

Arguments

input	(character) One of the following: **A character vector of sequences** See the example below for what this looks like. The parser read_fasta produces output like this. **A list of character vectors** Each vector should have one base per element. **A "DNAbin" object** This is the result of parsers like read.FASTA. **A list of "SeqFastadna" objects** This is the result of parsers like read.fasta. Either "input" or "file" must be supplied but not both.
file	The path to a FASTA file containing sequences to use. Either "input" or "file" must be supplied but not both.
include_seqs	(logical of length 1) If TRUE, include sequences in the output object.
add_species	(logical of length 1) If TRUE, add the species information to the taxonomy. In this databse, the species name often contains other information as well.

Details

The input file has a format like:

```
>S000448483 Sparassis crispa; MBUH-PIRJO&ILKKA94-1587/ss5 Lineage=Root;rootrank;Fun...
ggattcccctagtaactgcgagtgaagcgggaagagctcaaatttaaaatctggcggcgtcctcgtcgtccgagttgtaa
tctggagaagcgacatccgcgctggaccgtgtacaagtctcttggaaaagagcgtcgtagagggtgacaatcccgtcttt
...
```

Value

taxmap

See Also

Other parsers: parse_greengenes, parse_mothur_tax_summary, parse_mothur_taxonomy, parse_newick, parse_phyloseq, parse_phylo, parse_qiime_biom, parse_silva_fasta, parse_unite_general

parse_silva_fasta *Parse SILVA FASTA release*

Description

Parses an SILVA FASTA file that can be found at https://www.arb-silva.de/no_cache/download/archive/release_128/Exports/.

Usage

```
parse_silva_fasta(file = NULL, input = NULL, include_seqs = TRUE)
```

Arguments

file	The path to a FASTA file containing sequences to use. Either "input" or "file" must be supplied but not both.
input	(character) One of the following: **A character vector of sequences** See the example below for what this looks like. The parser read_fasta produces output like this. **A list of character vectors** Each vector should have one base per element. **A "DNAbin" object** This is the result of parsers like read.FASTA. **A list of "SeqFastadna" objects** This is the result of parsers like read.fasta. Either "input" or "file" must be supplied but not both.
include_seqs	(logical of length 1) If TRUE, include sequences in the output object.

Details

The input file has a format like:

```
 >GCVF01000431.1.2369
Bacteria;Proteobacteria;Gammaproteobacteria;Oceanospiril...
CGUGCACGGUGGAUGCCUUGGCAGCCAGAGGCGAUGAAGGACGUUGUAGCCUGCGAUAAGCUCCGGUUAGGUGGCAAACA
ACCGUUUGACCCGGAGAUCUCCGAAUGGGGCAACCCACCCGUUGUAAGGCGGGUAUCACCGACUGAAUCCAUAGGUCGGU
...
```

Value

taxmap

See Also

Other parsers: parse_greengenes, parse_mothur_tax_summary, parse_mothur_taxonomy, parse_newick, parse_phyloseq, parse_phylo, parse_qiime_biom, parse_rdp, parse_unite_general

`parse_unite_general` *Parse UNITE general release FASTA*

Description

Parse the UNITE general release FASTA file

Usage

```
parse_unite_general(input = NULL, file = NULL, include_seqs = TRUE)
```

Arguments

`input`	(`character`) One of the following: **A character vector of sequences** See the example below for what this looks like. The parser `read_fasta` produces output like this. **A list of character vectors** Each vector should have one base per element. **A "DNAbin" object** This is the result of parsers like `read.FASTA`. **A list of "SeqFastadna" objects** This is the result of parsers like `read.fasta`. Either "input" or "file" must be supplied but not both.
`file`	The path to a FASTA file containing sequences to use. Either "input" or "file" must be supplied but not both.
`include_seqs`	(`logical` of length 1) If `TRUE`, include sequences in the output object.

Details

The input file has a format like:

```
>Glomeromycota_sp|KJ484724|SH523877.07FU|reps|k__Fungi;p__Glomeromycota;c__unid...
ATAATTTGCCGAACCTAGCGTTAGCGCGAGGTTCTGCGATCAACACTTATATTTAAAACCCAACTCTTAAATTTTGTAT...
```

Value

`taxmap`

See Also

Other parsers: `parse_greengenes`, `parse_mothur_tax_summary`, `parse_mothur_taxonomy`, `parse_newick`, `parse_phyloseq`, `parse_phylo`, `parse_qiime_biom`, `parse_rdp`, `parse_silva_fasta`

`primersearch` *Use EMBOSS primersearch for in silico PCR*

Description

A pair of primers are aligned against a set of sequences. The location of the best hits, quality of match, and predicted amplicons are returned. Requires the EMBOSS tool kit (http://emboss.sourceforge.net/) to be installed.

Usage

```
primersearch(input = NULL, file = NULL, forward, reverse, mismatch = 5)
```

Arguments

`input`	(`character`) One of the following: **A character vector of sequences** See the example below for what this looks like. The parser `read_fasta` produces output like this. **A list of character vectors** Each vector should have one base per element. **A "DNAbin" object** This is the result of parsers like `read.FASTA`. **A list of "SeqFastadna" objects** This is the result of parsers like `read.fasta`. Either "input" or "file" must be supplied but not both.
`file`	The path to a FASTA file containing sequences to use. Either "input" or "file" must be supplied but not both.
`forward`	(`character` of length 1) The forward primer sequence
`reverse`	(`character` of length 1) The reverse primer sequence
`mismatch`	An integer vector of length 1. The percentage of mismatches allowed.

Details

It can be confusing how the primer sequence relates to the binding sites on a reference database sequence. A simplified diagram can help. For example, if the top strand below (5' -> 3') is the database sequence, the forward primer has the same sequence as the target region, since it will bind to the other strand (3' -> 5') during PCR and extend on the 3' end. However, the reverse primer must bind to the database strand, so it will have to be the complement of the reference sequence. It also has to be reversed to make it in the standard 5' -> 3' orientation. Therefore, the reverse primer must be the reverse complement of its binding site on the reference sequence.

```
Primer 1: 5' AAGTACCTTAACGGAATTATAG 3'
Primer 2: 5' GCTCCACCTACGAAACGAAT   3'

                                <- TAAGCAAAGCATCCACCTCG 5'
5' ...AAGTACCTTAACGGAATTATAG......ATTCGTTTCGTAGGTGGAGC... 3'

3' ...TTCATGGAATTGCCTTAATATC......TAAGCAAAGCATCCACCTCG... 5'
   5' AAGTACCTTAACGGAATTATAG ->
```

However, a database might have either the top or the bottom strand as a reference sequence. Since one implies the sequence of the other, either is valid, but this is another source of confusion. If we take the diagram above and rotate it 180 degrees, it would mean the same thing, but which primer we would want to call "forward" and which we would want to call "reverse" would change. Databases of a single locus (e.g. Greengenes) will likely have a convention for which strand will be present, so relative to this convention, there is a distinct "forward" and "reverse". However, computers dont know about this convention, so the "forward" primer is whichever primer has the same sequence as its binding region in the database (as opposed to the reverse complement). For this reason, primersearch will redefine which primer is "forward" and which is "reverse" based on how it binds the reference sequence. See the example code for a demonstration of this.

Value

A table with one row per predicted amplicon with the following info:

```
          (f_primer)
   5' AAGTACCTTAACGGAATTATAG ->        (r_primer)
                               <- TAAGCAAAGCATCCACCTCG 5'
5' ...AAGTACCTTAACGGAATTATAG......ATTCGTTTCGTAGGTGGAGC... 3'
      ^                     ^      ^                   ^
   f_start              f_end   r_rtart             r_end

      |--------------------||----||------------------|
             f_match       amplicon       r_match
      |-----------------------------------------------|
                           product


f_mismatch: The number of mismatches on the forward primer
r_mismatch: The number of mismatches on the reverse primer
input: The index of the input sequence
```

Installing EMBOSS

The command-line tool "primersearch" from the EMBOSS tool kit is needed to use this function. How you install EMBOSS will depend on your operating system:

Linux:

Open up a terminal and type:

```
sudo apt-get install emboss
```

Mac OSX:

The easiest way to install EMBOSS on OSX is to use homebrew. After installing homebrew, open up a terminal and type:

```
brew install homebrew/science/emboss
```

Windows:

There is an installer for Windows here:

ftp://emboss.open-bio.org/pub/EMBOSS/windows/mEMBOSS-6.5.0.0-setup.exe

Examples

```
## Not run:

### Dummy test data set ###

primer_1_site <- "AAGTACCTTAACGGAATTATAG"
primer_2_site <- "ATTCGTTTCGTAGGTGGAGC"
amplicon <- "NNNAGTGGATAGATAGGGGTTCTGTGGCGTTTGGGAATTAAAGATTAGAGANNN"
seq_1 <- paste0("AA", primer_1_site, amplicon, primer_2_site, "AAAA")
seq_2 <- rev_comp(seq_1)
f_primer <- "ACGTACCTTAACGGAATTATAG" # Note the "C" mismatch at position 2
r_primer <- rev_comp(primer_2_site)
seqs <- c(a = seq_1, b = seq_2)

result <- primersearch(seqs, forward = f_primer, reverse = r_primer)


### Real data set ###

# Get example FASTA file
fasta_path <- system.file(file.path("extdata", "silva_subset.fa"),
                          package = "metacoder")

# Parse the FASTA file as a taxmap object
obj <- parse_silva_fasta(file = fasta_path)

# Simulate PCR with primersearch
pcr_result <- primersearch(obj$data$tax_data$silva_seq,
                           forward = c("U519F" = "CAGYMGCCRCGGKAAHACC"),
                           reverse = c("Arch806R" = "GGACTACNSGGGTMTCTAAT"),
                           mismatch = 10)

# Add result to input table
#  NOTE: We want to add a function to handle running pcr on a
#        taxmap object directly, but we are still trying to figure out
#        the best way to implement it. For now, do the following:
obj$data$pcr <- pcr_result
obj$data$pcr$taxon_id <- obj$data$tax_data$taxon_id[pcr_result$input]

# Visualize which taxa were amplified
#  This work because only amplicons are returned by `primersearch`
n_amplified <- obj$obs_apply("pcr",
                             function(x) length(unique(x)),
                             value = "input",
                             simplify = TRUE)
prop_amped <- n_amplified / obj$n_obs()
heat_tree(obj,
          node_label = taxon_names,
          node_color = prop_amped,
          node_color_range = c("grey", "red", "purple", "green"),
          node_color_trans = "linear",
          node_color_axis_label = "Proportion amplified",
```

```
            node_size = n_obs,
            node_size_axis_label = "Number of sequences",
            layout = "da",
            initial_layout = "re")


## End(Not run)
```

`qualitative_palette` *The default qualitative color palette*

Description

Returns the default color palette for qualitative data

Usage

```
qualitative_palette()
```

Value

`character` of hex color codes

Examples

```
qualitative_palette()
```

`quantative_palette` *The default quantative color palette*

Description

Returns the default color palette for quantative data.

Usage

```
quantative_palette()
```

Value

`character` of hex color codes

Examples

```
quantative_palette()
```

`rarefy_obs` *Calculate rarefied observation counts*

Description

For a given table in a `taxmap` object, rarefy counts to a constant total. This is a wrapper around `rrarefy` that automatically detects which columns are numeric and handles the reformatting needed to use tibbles.

Usage

```
rarefy_obs(obj, dataset, sample_size = NULL, cols = NULL,
  other_cols = FALSE, out_names = NULL)
```

Arguments

`obj`	A `taxmap` object
`dataset`	The name of a table in `obj` that contains counts.
`sample_size`	The sample size counts will be rarefied to. This can be either a single integer or a vector of integers of equal length to the number of columns.
`cols`	The names/indexes of columns in `dataset` to use. By default, all numeric columns are used. Takes one of the following inputs: **TRUE/FALSE:** All/No columns will used. **Character vector:** The names of columns to use **Numeric vector:** The indexes of columns to use **Vector of TRUE/FALSE of length equal to the number of columns:** Use the columns corresponding to `TRUE` values.
`other_cols`	Preserve in the output non-target columns present in the input data. New columns with proportions will always be on the end. The "taxon_id" column will always be preserved in the front. Takes one of the following inputs: **TRUE/FALSE:** All non-target columns will be preserved or not. **Character vector:** The names of columns to preserve **Numeric vector:** The indexes of columns to preserve **Vector of TRUE/FALSE of length equal to the number of columns:** Preserve the columns corresponding to `TRUE` values.
`out_names`	If supplied, rename the output proportion columns. Must be the same length as `cold`.

Value

A tibble

See Also

Other calculations: calc_n_samples, calc_obs_props, calc_taxon_abund, compare_groups, zero_low_counts

Examples

```
## Not run:
# Parse dataset for examples
x = parse_tax_data(hmp_otus, class_cols = "lineage", class_sep = ";",
                   class_key = c(tax_rank = "info", tax_name = "taxon_name"),
                   class_regex = "^(.+)__(.+)$")

# Rarefy all numeric columns
rarefy_obs(x, "tax_data")

# Rarefy a subset of columns
rarefy_obs(x, "tax_data", cols = c("700035949", "700097855", "700100489"))
rarefy_obs(x, "tax_data", cols = 4:6)
rarefy_obs(x, "tax_data", cols = startsWith(colnames(x$data$tax_data), "70001"))

# Including all other columns in ouput
rarefy_obs(x, "tax_data", other_cols = TRUE)

# Inlcuding specific columns in output
rarefy_obs(x, "tax_data", cols = c("700035949", "700097855", "700100489"),
               other_cols = 2:3)

# Rename output columns
rarefy_obs(x, "tax_data", cols = c("700035949", "700097855", "700100489"),
               out_names = c("a", "b", "c"))


## End(Not run)
```

---

read_fasta *Read a FASTA file*

Description

Reads a FASTA file. This is the FASTA parser for metacoder. It simply tries to read a FASTA file into a named character vector with minimal fuss. It does not do any checks for valid characters etc. Other FASTA parsers you might want to consider include read.FASTA or read.fasta.

Usage

```
read_fasta(file_path)
```

Arguments

file_path	(character of length 1) The path to a file to read.

Value

named character vector

Examples

```
# Get example FASTA file
fasta_path <- system.file(file.path("extdata", "silva_subset.fa"),
                          package = "metacoder")

# Read fasta file
my_seqs <- read_fasta(fasta_path)
```

---

reverse *Reverse sequences*

Description

Find the reverse of one or more sequences stored as a character vector. This is a wrapper for rev for character vectors instead of lists of character vectors with one value per letter.

Usage

```
reverse(seqs)
```

Arguments

seqs	A character vector with one element per sequence.

See Also

Other sequence transformations: complement, rev_comp

Examples

```
reverse(c("aagtgGGTGaa", "AAGTGGT"))
```

rev_comp *Revere complement sequences*

Description

Make the reverse complement of one or more sequences stored as a character vector. This is a wrapper for comp for character vectors instead of lists of character vectors with one value per letter. IUPAC ambiguity codes are handled and the upper/lower case is preserved.

Usage

```
rev_comp(seqs)
```

Arguments

seqs	A character vector with one element per sequence.

See Also

Other sequence transformations: complement, reverse

Examples

```
rev_comp(c("aagtgGGTGaa", "AAGTGGT"))
```

write_greengenes *Write an imitation of the Greengenes databse*

Description

Attempts to save taxonomic and sequence information of a taxmap object in the Greengenes output format. If the taxmap object was created using parse_greengenes, then it should be able to replicate the format exactly with the default settings.

Usage

```
write_greengenes(obj, tax_file = NULL, seq_file = NULL,
  tax_names = obj$get_data("taxon_names")[[1]],
  ranks = obj$get_data("gg_rank")[[1]], ids = obj$get_data("gg_id")[[1]],
  sequences = obj$get_data("gg_seq")[[1]])
```

Arguments

obj	A taxmap object
tax_file	(character of length 1) The file path to save the taxonomy file.
seq_file	(character of length 1) The file path to save the sequence fasta file. This is optional.
tax_names	(character named by taxon ids) The names of taxa
ranks	(character named by taxon ids) The ranks of taxa
ids	(character named by taxon ids) Sequence ids
sequences	(character named by taxon ids) Sequences

Details

The taxonomy output file has a format like:

```
228054  k__Bacteria; p__Cyanobacteria; c__Synechococcophycideae; o__Synech...
844608  k__Bacteria; p__Cyanobacteria; c__Synechococcophycideae; o__Synech...
...
```

The optional sequence file has a format like:

```
>1111886
AACGAACGCTGGCGGCATGCCTAACACATGCAAGTCGAACGAGACCTTCGGGTCTAGTGGCGCACGGGTGCGTA...
>1111885
AGAGTTTGATCCTGGCTCAGAATGAACGCTGGCGGCGTGCCTAACACATGCAAGTCGTACGAGAAATCCCGAGC...
...
```

See Also

Other writers: `write_mothur_taxonomy`, `write_rdp`, `write_unite_general`

---

write_mothur_taxonomy *Write an imitation of the Mothur taxonomy file*

Description

Attempts to save taxonomic information of a taxmap object in the mothur '*.taxonomy' format. If the taxmap object was created using `parse_mothur_taxonomy`, then it should be able to replicate the format exactly with the default settings.

Usage

```
write_mothur_taxonomy(obj, file, tax_names = obj$get_data("taxon_names")[[1]],
  ids = obj$get_data("sequence_id")[[1]],
  scores = obj$get_data("score")[[1]])
```

Arguments

obj	A taxmap object
file	(character of length 1) The file path to save the sequence fasta file. This is optional.
tax_names	(character named by taxon ids) The names of taxa
ids	(character named by taxon ids) Sequence ids
scores	TBD

Details

The output file has a format like:

```
AY457915 Bacteria(100);Firmicutes(99);Clostridiales(99);Johnsone...
AY457914 Bacteria(100);Firmicutes(100);Clostridiales(100);Johnso...
AY457913 Bacteria(100);Firmicutes(100);Clostridiales(100);Johnso...
AY457912 Bacteria(100);Firmicutes(99);Clostridiales(99);Johnsone...
AY457911 Bacteria(100);Firmicutes(99);Clostridiales(98);Ruminoco...
```

or...

```
AY457915 Bacteria;Firmicutes;Clostridiales;Johnsonella_et_rel.;J...
AY457914 Bacteria;Firmicutes;Clostridiales;Johnsonella_et_rel.;J...
AY457913 Bacteria;Firmicutes;Clostridiales;Johnsonella_et_rel.;J...
AY457912 Bacteria;Firmicutes;Clostridiales;Johnsonella_et_rel.;J...
AY457911 Bacteria;Firmicutes;Clostridiales;Ruminococcus_et_rel.;...
```

See Also

Other writers: write_greengenes, write_rdp, write_unite_general

---

write_rdp *Write an imitation of the RDP FASTA databse*

Description

Attempts to save taxonomic and sequence information of a taxmap object in the RDP FASTA format. If the taxmap object was created using parse_rdp, then it should be able to replicate the format exactly with the default settings.

Usage

```
write_rdp(obj, file, tax_names = obj$get_data("taxon_names")[[1]],
  ranks = obj$get_data("rdp_rank")[[1]], ids = obj$get_data("rdp_id")[[1]],
  info = obj$get_data("seq_name")[[1]],
  sequences = obj$get_data("rdp_seq")[[1]])
```

Arguments

obj	A taxmap object
file	(character of length 1) The file path to save the sequence fasta file. This is optional.
tax_names	(character named by taxon ids) The names of taxa
ranks	(character named by taxon ids) The ranks of taxa
ids	(character named by taxon ids) Sequence ids
info	(character named by taxon ids) Info associated with sequences. In the example output shown here, this field corresponds to "Sparassis crispa; MBUH-PIRJO&ILKKA94-1587/ss5"
sequences	(character named by taxon ids) Sequences

Details

The output file has a format like:

```
>S000448483 Sparassis crispa; MBUH-PIRJO&ILKKA94-1587/ss5 Lineage=Root;rootrank;Fun...
ggattcccctagtaactgcgagtgaagcgggaagagctcaaatttaaaatctggcggcgtcctcgtcgtccgagttgtaa
tctggagaagcgacatccgcgctggaccgtgtacaagtctcttggaaaagagcgtcgtagagggtgacaatcccgtcttt
...
```

See Also

Other writers: write_greengenes, write_mothur_taxonomy, write_unite_general

---

write_silva_fasta *Write an imitation of the SILVA FASTA databse*

Description

Attempts to save taxonomic and sequence information of a taxmap object in the SILVA FASTA format. If the taxmap object was created using parse_silva_fasta, then it should be able to replicate the format exactly with the default settings.

Usage

```
write_silva_fasta(obj, file, tax_names = obj$get_data("taxon_names")[[1]],
  other_names = obj$get_data("other_name")[[1]],
  ids = obj$get_data("ncbi_id")[[1]],
  start = obj$get_data("start_pos")[[1]],
  end = obj$get_data("end_pos")[[1]],
  sequences = obj$get_data("silva_seq")[[1]])
```

Arguments

obj	A taxmap object
file	(character of length 1) The file path to save the sequence fasta file. This is optional.
tax_names	(character named by taxon ids) The names of taxa
other_names	(character named by taxon ids) Alternate names of taxa. Will be added after the primary name.
ids	(character named by taxon ids) Sequence ids
start	(character) The start position of the sequence.
end	(character) The end position of the sequence.
sequences	(character named by taxon ids) Sequences

Details

The output file has a format like:

```
>GCVF01000431.1.2369 Bacteria;Proteobacteria;Gammaproteobacteria;Oceanospiril...
CGUGCACGGUGGAUGCCUUGGCAGCCAGAGGCGAUGAAGGACGUUGUAGCCUGCGAUAAGCUCCGGUUAGGUGGCAAACA
ACCGUUUGACCCGGAGAUCUCCGAAUGGGGCAACCCACCCGUUGUAAGGCGGGUAUCACCGACUGAAUCCAUAGGUCGGU
...
```

`write_unite_general` *Write an imitation of the UNITE general FASTA databse*

Description

Attempts to save taxonomic and sequence information of a taxmap object in the UNITE general FASTA format. If the taxmap object was created using `parse_unite_general`, then it should be able to replicate the format exactly with the default settings.

Usage

```
write_unite_general(obj, file, tax_names = obj$get_data("taxon_names")[[1]],
  ranks = obj$get_data("unite_rank")[[1]],
  sequences = obj$get_data("unite_seq")[[1]],
  seq_name = obj$get_data("organism")[[1]],
  ids = obj$get_data("unite_id")[[1]],
  gb_acc = obj$get_data("acc_num")[[1]],
  type = obj$get_data("unite_type")[[1]])
```

Arguments

obj	A taxmap object
file	(character of length 1) The file path to save the sequence fasta file. This is optional.
tax_names	(character named by taxon ids) The names of taxa
ranks	(character named by taxon ids) The ranks of taxa
sequences	(character named by taxon ids) Sequences
seq_name	(character named by taxon ids) Name of sequences. Usually a taxon name.
ids	(character named by taxon ids) UNITE sequence ids
gb_acc	(character named by taxon ids) Genbank accession numbers
type	(character named by taxon ids) What type of sequence it is. Usually "rep" or "ref".

Details

The output file has a format like:

```
>Glomeromycota_sp|KJ484724|SH523877.07FU|reps|k__Fungi;p__Glomeromycota;c__unid...
ATAATTTGCCGAACCTAGCGTTAGCGCGAGGTTCTGCGATCAACACTTATATTTAAAACCCAACTCTTAAATTTTGTAT...
...
```

See Also

Other writers: `write_greengenes`, `write_mothur_taxonomy`, `write_rdp`

---

zero_low_counts *Replace low counts with zero*

Description

For a given table in a `taxmap` object, convert all counts below a minimum number to zero. This is useful for effectively removing "singletons", "doubletons", or other low abundance counts.

Usage

```
zero_low_counts(obj, dataset, min_count = 2, use_total = FALSE,
  cols = NULL, other_cols = FALSE, out_names = NULL)
```

Arguments

obj	A taxmap object
dataset	The name of a table in obj that contains counts.
min_count	The minimum number of counts needed for a count to remain unchanged. Any could less than this will be converted to a zero. For example, min_count = 2 would remove singletons.
use_total	If TRUE, the min_count applies to the total count for each row (e.g. OTU counts for all samples), rather than each cell in the table. For example use_total = TRUE, min_count = 10 would convert all counts of any row to zero if the total for all counts in that row was less than 10.
cols	The names/indexes of columns in dataset to use. By default, all numeric columns are used. Takes one of the following inputs: **TRUE/FALSE:** All/No columns will used. **Character vector:** The names of columns to use **Numeric vector:** The indexes of columns to use **Vector of TRUE/FALSE of length equal to the number of columns:** Use the columns corresponding to TRUE values.
other_cols	Preserve in the output non-target columns present in the input data. New columns with proportions will always be on the end. The "taxon_id" column will always be preserved in the front. Takes one of the following inputs: **TRUE/FALSE:** All non-target columns will be preserved or not. **Character vector:** The names of columns to preserve **Numeric vector:** The indexes of columns to preserve **Vector of TRUE/FALSE of length equal to the number of columns:** Preserve the columns corresponding to TRUE values.
out_names	If supplied, rename the output proportion columns. Must be the same length as cold.

Value

A tibble

See Also

Other calculations: calc_n_samples, calc_obs_props, calc_taxon_abund, compare_groups, rarefy_obs

Examples

```
## Not run:
# Parse dataset for examples
x = parse_tax_data(hmp_otus, class_cols = "lineage", class_sep = ";",
                   class_key = c(tax_rank = "info", tax_name = "taxon_name"),
                   class_regex = "^(.+)__(.+)$")

# Calculate proportions for all numeric columns
```

```
calc_obs_props(x, "tax_data")

# Calculate proportions for a subset of columns
calc_obs_props(x, "tax_data", cols = c("700035949", "700097855", "700100489"))
calc_obs_props(x, "tax_data", cols = 4:6)
calc_obs_props(x, "tax_data", cols = startsWith(colnames(x$data$tax_data), "70001"))

# Including all other columns in ouput
calc_obs_props(x, "tax_data", other_cols = TRUE)

# Inlcuding specific columns in output
calc_obs_props(x, "tax_data", cols = c("700035949", "700097855", "700100489"),
               other_cols = 2:3)

# Rename output columns
calc_obs_props(x, "tax_data", cols = c("700035949", "700097855", "700100489"),
               out_names = c("a", "b", "c"))


## End(Not run)
```

Index