Package: microclass (via r-universe)

September 8, 2024

Encoding UTF-8

Type Package

Title Tools for taxonomic classification of prokaryotes

Version 1.2

Date 2023-08-21

Author Kristian Hovde Liland, Hilde Vinje, Lars Snipen

Maintainer Lars Snipen <lars.snipen@nmbu.no>

Description Functions for working with taxonomic classifications in R

License GPL (>= 2)

Depends R (>= 4.0.0), microseq, microcontax, data.table, dplyr, stringr, rlang, Matrix

Imports Rcpp (>= 0.11.1), RcppParallel, tibble

LinkingTo Rcpp (>= 0.11.1), RcppEigen, RcppParallel

SystemRequirements GNU make

RoxygenNote 7.2.3

Suggests knitr, rmarkdown

VignetteBuilder knitr

Repository https://larssnip.r-universe.dev

RemoteUrl https://github.com/larssnip/microclass

RemoteRef HEAD

RemoteSha 3f3ac0ad10650957168e1d9d903e8eea138aae36

Contents

microclass-package 2
blastClassify16S 3
blastDbase16S 4
bracken_read_report 5
branch_list2qiime 6

branch_list2table . . . 7
branch_prune . . . 8
branch_retrieve . . . 8
branch_taxid2name . . . 9
clr . . . 10
getDomain . . . 11
KmerCount . . . 12
kraken2_read_report . . . 13
kraken2_read_table . . . 14
multinomClassify . . . 15
multinomTrain . . . 17
rdpClassify . . . 18
rdpTrain . . . 19
read_names_dmp . . . 20
read_nodes_dmp . . . 21
setParallel . . . 22
small.16S . . . 23
subset_clade . . . 23
subset_tree . . . 24
taxMachine . . . 25

Index **27**

`microclass-package` *Methods for 16S based taxonomic classification of prokaryotes*

Description

The package provides functions for assigning 16S sequence data to a taxonomic level in the tree-of-life for prokaryotes.

Usage

```
microclass()
```

Details

Package:	microclass
Type:	Package
Version:	1.2
Date:	2023-08-21
License:	GPL-2

Author(s)

Hilde Vinje, Kristian Hovde Liland, Lars Snipen.
Maintainer: Lars Snipen <lars.snipen@nmbu.no>

`blastClassify16S` *Classifying using BLAST*

Description

A 16S based classification based on BLAST.

Usage

```
blastClassify16S(sequence, bdb)
```

Arguments

`sequence`	Character vector of 16S sequences to classify.
`bdb`	Name of BLAST data base, see `blastDbase16S`.

Details

A vector of 16S sequences (DNA) are classified by first using BLAST `blastn` against a database of 16S DNA sequences, and then classify according to the nearest-neighbour principle. The nearest neighbour of a query sequence is the hit with the largest bitscore. The blast+ software https://blast.ncbi.nlm.nih.gov/Blast.cgi?PAGE_TYPE=BlastDocs&DOC_TYPE=Download must be installed on the system. Type `system("blastn -help")` in the Console window, and a sensible Help-text should appear.

The database must contain 16S sequences where the Header starts with a token specifying the taxon. More specifically, the tokens must look like:

<taxon>_1

<taxon>_2

...etc

where <taxon> is some proper taxon name. Use `blastDbase16S` to make such databases.

The identity of each alignment is also computed. This should be close to 1.0 for a classification to be trusted. Identity values below 0.95 could indicate uncertain classifications, but this will vary between taxa.

Value

A `data.frame` with two columns: Taxon is the predicted taxon for each `sequence` and Identity is the corresponding identity-value. If no BLAST hit is seen, the sequence is `"unclassified"`.

Author(s)

Lars Snipen.

See Also

blastDbase16S.

Examples

```
data("small.16S")
## Not run:
dbase <- blastDbase16S("test", small.16S$Sequence, word(small.16S$Header, 2, 2))
reads <- str_sub(small.16S$Sequence, 100, 550)
blastClassify16S(reads, dbase) %>%
  bind_cols(small.16S) -> tbl

## End(Not run)
```

---

blastDbase16S *Building a BLAST database*

Description

Building a BLAST database for 16S based classification.

Usage

```
blastDbase16S(name, sequence, taxon)
```

Arguments

name	The name of the database (text).
sequence	A character vector with 16S sequence data.
taxon	A character vector with taxon information.

Details

This functions builds a database using the makeblastdb program of the BLAST+ software https://blast.ncbi.nlm.nih.gov/Blast.cgi?PAGE_TYPE=BlastDocs&DOC_TYPE=Download. Thus, this software must be available on the system when using this function. If you type system("makeblastdb -help") in the Console window some meaningful Help-text should be displayed.

This function is most typically used prior to blastClassify16S to set up the database before searching and classifying. It can be seen as the 'training step' of a BLAST-based classification procedure.

The sequence must be a vector of DNA-sequences (16S sequences). The taxon is a vector of the same length as sequence, containing the correpsonding taxon information.

Value

The database files are created, and the name of the database (name) is returned.

Author(s)

Lars Snipen.

See Also

blastClassify16S.

Examples

```
# See examples for blastClassify16S.
```

---

bracken_read_report *Reads bracken reports*

Description

Reads the report output from bracken.

Usage

```
bracken_read_report(report.file)
```

Arguments

report.file	name of file with bracken report output.

Details

The report output from bracken is a table with one row for each taxon with assigned reads. This is the file produced by bracken using the -o option.

Note that by the -w option bracken will output a report in kraken2 format, see kraken2_read_report.

Value

A data.frame with the bracken results for each taxon. It has the columns:

- name. The name of this taxon in the taxonomy.
- tax_id. The taxonomy identifier of this taxon in the taxonomy.
- rank. The rank of this taxon in the taxonomy.
- tax_count_krk. The number of reads classified to this taxon by kraken2.
- added_brk. The additional reads assigned by bracken.
- tax_count. The final number of reads assigned to this taxon.
- fraction. The fraction of the total number of reads assigned to this taxon.

Author(s)

Lars Snipen.

See Also

kraken2_read_report.

`branch_list2qiime` *Branch list to QIIME format*

Description

Creates QIIME formatted texts from a branch list.

Usage

```
branch_list2qiime(branch.lst)
```

Arguments

`branch.lst`	a list of branches.

Details

A QIIME formatted branch is a text with the format

`"d__domain;p__phylum;c__class;o__order;f__family;g__genus;s__species"`

where the words domain, phylum, etc are replaced by taxon names. This function converts a list of N branches to N such texts.

NOTE 1: This is only meaningful if the `branch.lst` contains names instead of tax_id's, see branch_taxid2name.

NOTE 2: The ranks are fixed to the ones listed above, where domain is the same as superkingdom in the NCBI taxonomy.

Value

The vector of texts, one for each element in `branch.lst`.

Author(s)

Lars Snipen.

`branch_list2table` *Branch list na dtable conversion*

Description

Turns a list of branches into a table with, or vice versa.

Usage

```
branch_list2table(
  branch.lst,
 ranks = c("superkingdom", "phylum", "class", "order", "family", "genus", "species")
)
```

Arguments

`branch.lst`	a list of branches.
`ranks`	texts specifying the ranks to keep and their ordering.
`branch.tbl`	a table of branches.

Details

Instead of having branches in a list, it is convenient in R to have data in tables. The `branch_list2table` converts a list of branches into a table, with one row for each branch and a column for each rank. Since branches may contain different ranks, they must first be pruned to have the exact same ranks in the exact same order. This is done by `branch_prune` inside this function. The ranks specified in `ranks` are the columns in the table returned here.

The function `branch_table2list` converts back again, i.e. from a table to a list.

Value

The `branch_list2table` returns a tibble with one row for each branch and a column for each rank. The cells contain the tax_id's or names in the `branch.lst`.

The `branch_table2list` returns a list where each element contains a named vector of tax_id or texts, and the name of each element is its rank in the taxonomy.

Author(s)

Lars Snipen.

branch_prune *Filtering ranks in branches*

Description

Keeps only specified ranks in specified ordering in all branches.

Usage

```
branch_prune(
  branch.lst,
 ranks = c("superkingdom", "phylum", "class", "order", "family", "genus", "species")
)
```

Arguments

branch.lst	a list of branches.
ranks	texts specifying the ranks to keep and their ordering.

Details

Branches in the taxonomy tree may have many different ranks or levels. This function is used to prune them all to have the same set of ranks. Ranks may be missing in some branches.This function will then fill in NA in these cells, ensuring all branches have the exact same ranks in the exact same ordering. This is convenient for turning the list into a table, using as_tibble.

Value

A new branch-list with pruned branches all containing the exact same ranks.

Author(s)

Lars Snipen.

branch_retrieve *Retrieves branches*

Description

Retrieves branches in the taxonomy tree.

Usage

```
branch_retrieve(leaf_tax_id, nodes.dmp)
```

Arguments

`leaf.tax.id`	a vector of tax_id (integers) of the branch leaf/leaves.
`nodes.dmp`	a nodes.dmp table.

Details

This function retrieves the branch from the taxonomy tree that ends at the `leaf.tax.id` and starts at the root of tree defined in the table `nodes.dmp` (see `read_nodes_dmp`).

NB! Only the root node in nodes.dmp must have its own tax_id as parent_tax_id! This is the criterion for ending a branch.

Multiple leaves may be given as argument, resulting in multiple branches retrieved.

Value

A branch list, which is simply a list containing named vectors of tax_id integers, from the root to the leaf of each branch. The name of each element is its rank. You may replace the tax_id integers by the taxon names using `branch_taxid2name`.

Author(s)

Lars Snipen.

See Also

`branch_taxid2name`, `subset_clade`.

`branch_taxid2name` *Replace tax_id with name*

Description

Converts vectors of tax_id to vectors of names, or vice versa.

Usage

```
branch_taxid2name(branch.lst, names.dmp)
```

Arguments

`branch.lst`	a list of branches.
`names.dmp`	a names.dmp table.

Details

This function is used to convert the tax_id's to name_txt's in a branch-list, or vice versa. See `branch_retrieve` for more about branch-lists.

`branch_taxid2name`: The `names.dmp$name_txt` may contain many names for each tax_id, and the argument `name.class` is used to select the type of name to use from the column `names.dmp$name_class`.

`branch_name2taxid`: The matching of names is exact (using `match`), which means only the first occurrence of the name_txt is used. You are responsible for using a `names.dmp` with unique texts in `names.dmp$name_txt`.

See `read_names_dmp` for more about `names.dmp` tables.

Value

A list containing named vectors of either texts (`branch_taxid2name`) or integers (`branch_name2taxid`), from the root to the leaf of each branch. The name of each element is its rank.

Author(s)

Lars Snipen.

---

clr *Centred log-ratio transform*

Description

Transforms readcount-matrix with Aitchisons transform.

Usage

```
clr(readcount.mat, n.pseudo = 1)
```

Arguments

`readcount.mat`	matrix with readcount data.
`n.pseudo`	number of pseudo-readcounts to add.

Details

This is a standard implementation of the Aitchisons centered log-ratio transform (Aitchison 1986) for compositional data. Readcount data can be seen as compositional data since the total number of readcounts in a sample does not carry any information about the biology, but is simply an effect of sequencing depth. Thus, the information in the data lies in the relative values, not the absolute. By transforming such data with this function, you get data who are better suited for a number of downstream analyses, e.g. typically analyses making use of sum-of-squares type of statistics, like PCA, PLS, ANOVA or clustering with euclidean distances.

The `readcount.mat` must have the samples in the rows and the taxa in the columns. Transpose if necessary.

The transform does not accept zeros in any cell of the `readcount.mat`. To cope with this you add pseudo-counts. Bu default 1 additional read is assigned to all cells in `readcount.mat`. You may change this value, and it need not be an integer. The rationale behind this is that we a priori assume a uniform distribution of the taxa, and the more pseudo-counts you add, the more weight you give to this prior.

Value

A matrix of same size as the input, but with transformed readcounts.

Author(s)

Lars Snipen.

References

Aitchison J. The Statistical Analysis of Compositional Data. London, UK: Chapman & Hall; 1986.

`getDomain` *Extractor functions for QIIME taxonomy*

Description

Extracting taxonomic information from FASTA headers.

Usage

```
getDomain(header)
getPhylum(header)
getClass(header)
getOrder(header)
getFamily(header)
getGenus(header)
getTag(header)
getTaxonomy(header)
```

Arguments

`header`	A vector of texts, the `Header` column from reading a FASTA file, containing taxonomy information in the proper format.

Details

The ConTax data sets (see package `microcontx`) are FASTA files where the `Header` contains texts according to a strict format inherited from QIIME:

It always starts with a short text, a Tag, which is a unique identifier for every sequence. The function `getTag` will extract this from the `header`.

After the Tag follows one or more tokens. One of these tokens is a string with the following format, inherited from QIIME:

"k__<...>;p__<...>;c__<...>;o__<...>;f__<...>;g__<...>;"

where <...> is some proper text. Here is an example of a proper string:

"k__Bacteria;p__Firmicutes;c__Bacilli;o__Bacillales;f__Staphylococcaceae;g__Staphylococcus;"

The functions `getDomain`, ..., `getGenus` extracts the corresponding information from the `header`. `getTaxonomy` combines all taxonomy extractors, combines these in a table and imputes missing taxa with parent taxa.

Value

A vector containing the sub-texts extracted from each `header` text, but `getTaxonomy` returns a table with the full taxonomy, one row for each input `header`

Author(s)

Lars Snipen.

See Also

`contax.trim`, `medoids`.

Examples

```
data(contax.trim)
getTag(contax.trim$Header)
getGenus(contax.trim$Header)
getPhylum(contax.trim$Header)
```

KmerCount *K-mer counting*

Description

Counting overlapping words of length K in DNA/RNA sequences.

Usage

```
KmerCount(sequences, K = 1, col.names = FALSE)
```

Arguments

`sequences`	Vector of sequences (text).
`K`	Word length (integer).
`col.names`	Logical indicating if the words should be added as columns names.

Details

For each input sequence, the frequency of every word of length K is counted. Counting is done with overlap. The counting itself is done by a C++ function.

With col.names = TRUE the K-mers are added as column names, but this makes the computations slower.

Value

A matrix with one row for each sequence in sequences and one column for each possible word of lengthK.

Author(s)

Kristian Hovde Liland and Lars Snipen.

See Also

multinomTrain, multinomClassify.

Examples

```
KmerCount("ATGCCTGAACTGACCTGC", K = 2)
```

---

kraken2_read_report *Reads* kraken2 *reports*

Description

Reads the report output from kraken2.

Usage

```
kraken2_read_report(report.file)
```

Arguments

report.file	name of file with kraken2 report output.

Details

The report output from kraken2 is a table with one row for each taxon detected by kraken2. Run kraken2 using the --report option to get this convenient summary table.

Note that the software bracken may also output a report in this format.

Value

A data.frame with the kraken2 results for each taxon. It has the columns:

- percent. The percentage of the reads classified to this taxon.
- clade_count. The total number of reads classified to the clade originating from this taxon.
- tax_count. The number of reads assigned directly to this taxon.
- rank. The rank of this taxon in the taxonomy.
- tax_id. The taxonomy identifier of this taxon in the taxonomy.
- name. The name of this taxon in the taxonomy.

Author(s)

Lars Snipen.

See Also

kraken2_read_table, bracken_read_report.

---

kraken2_read_table *Reads raw* kraken2 *output*

Description

Reads the raw output table from kraken2.

Usage

```
kraken2_read_table(krk.file, filter = TRUE)
```

Arguments

krk.file	name of file with raw kraken2 output.
filter.	returns only classified reads if TRUE.

Details

The raw output from kraken2 is a huge table with one row for each read classified.

Note that in most cases only the report, produced by running kraken2 with the --report otpion, is what you want. Then use the function kraken2_read_report.

Value

A data.frame with the kraken2 results for each read. It has the columns:

- status. C for classified reads, U for unclassified (if `filter = TRUE` only C).
- read_id. Short text identifying each read.
- name. Name of taxon, if classified.
- length. Length of read.
- tax_count. The k-mer counts for the read.

Author(s)

Lars Snipen.

See Also

`kraken2_read_report`.

`multinomClassify` *Classifying with a Multinomial model*

Description

Classifying sequences by a trained Multinomial model.

Usage

```
multinomClassify(
  sequence,
  multinom.prob,
  post.prob = FALSE,
  prior = FALSE,
  full.post.prob = FALSE
)
```

Arguments

`sequence`	Character vector of sequences to classify.
`multinom.prob`	A matrix of multinomial probabilities, see `multinomTrain`.
`post.prob`	Logical indicating if posterior log-probabilities should be returned.
`prior`	Logical indicating if classification should be done by flat priors (default) or with empirical priors.
`full.post.prob`	Logical indicating if full posterior probability matrix should be returned.

Details

The classification step of the multinomial method (Vinje et al, 2015) means counting K-mers on all sequences, and computing the posterior probabilities for each taxon given the trained model. The predicted taxon for each input sequence is the one with the maximum posterior probability for that sequence.

By setting `post.prob = TRUE` you will get the log-probability of the best and second best taxon for each sequence. This may be used for evaluating the certainty in the classifications.

The classification is parallelized through RcppParallel employing Intel TBB and TinyThread. By default all available processing cores are used. This can be changed using the function `setParallel`.

Value

If `post.prob = FALSE` a character vector of predicted taxa is returned.

If `post.prob = TRUE` a `data.frame` with three columns is returned.

- taxon. The predicted taxa, one for each sequence in `sequence`.
- post_prob. The posterior log-probability of the assigned taxon.
- post_prob_2. The largest posterior log-probability of the other taxa.

Author(s)

Kristian Hovde Liland and Lars Snipen.

References

Vinje, H, Liland, KH, Almøy, T, Snipen, L. (2015). Comparing K-mer based methods for improved classification of 16S sequences. BMC Bioinformatics, 16:205.

See Also

`KmerCount`, `multinomTrain`.

Examples

```
data("small.16S")
seq <- small.16S$Sequence
tax <- sapply(strsplit(small.16S$Header,split=" "),function(x){x[2]})
## Not run:
trn <- multinomTrain(seq,tax)
primer.515f <- "GTGYCAGCMGCCGCGGTAA"
primer.806rB <- "GGACTACNVGGGTWTCTAAT"
reads <- amplicon(seq, primer.515f, primer.806rB)
predicted <- multinomClassify(unlist(reads[nchar(reads)>0]),trn)
print(predicted)

## End(Not run)
```

`multinomTrain` *Training multinomial model*

Description

Training the multinomial K-mer method on sequence data.

Usage

```
multinomTrain(sequence, taxon, K = 5, col.names = FALSE, n.pseudo = 1)
```

Arguments

`sequence`	Character vector of sequences.
`taxon`	Character vector of taxon labels for each sequence.
`K`	Word length (integer).
`col.names`	Logical indicating if column names (K-mers) should be added to the trained model matrix.
`n.pseudo`	Number of pseudo-counts to use (positive numerics, need not be integer). Special case -1 will only return word counts, not log-probabilities.

Details

The training step of the multinomial method (Vinje et al, 2015) means counting K-mers on all sequences and compute their multinomial probabilities for each taxon. `n.pseudo` pseudo-counts are added equally to all K-mers, before probabilities are estimated. The optimal choice of `n.pseudo` will depend on `K` and the training data set.

Adding the actual K-mers as column names (`col.names = TRUE`) will slow down the computations.

The relative taxon frequencies in the `taxon` input are also computed and returned as an attribute to the probability matrix.

Value

A matrix with the multinomial probabilities, one row for each `taxon` and one column for each K-mer. The sum of each row is 1.0. No probabilities are 0 if `n.pseudo` > 0.0.

The matrix has an attribute `attr("prior",)`, that contains the relative taxon frequencies.

Author(s)

Kristian Hovde Liland and Lars Snipen.

References

Vinje, H, Liland, KH, Almøy, T, Snipen, L. (2015). Comparing K-mer based methods for improved classification of 16S sequences. BMC Bioinformatics, 16:205.

See Also

KmerCount, multinomClassify.

Examples

```
# See examples for multinomClassify
```

---

rdpClassify *Classifying with the RDP classifier*

Description

Classifying sequences by a trained presence/absence K-mer model.

Usage

```
rdpClassify(sequence, trained.model, post.prob = FALSE, prior = FALSE)
```

Arguments

sequence	Character vector of sequences to classify.
trained.model	A list with a trained model, see rdpTrain.
post.prob	Logical indicating if posterior log-probabilities should be returned.
prior	Logical indicating if classification should be done by flat priors (default) or with empirical priors (prior=TRUE).

Details

The classification step of the presence/absence method known as the RDP classifier (Wang et al 2007) means looking for K-mers on all sequences, and computing the posterior probabilities for each taxon using a trained model and a naive Bayes assumption. The predicted taxon is the one producing the maximum posterior probability, for each sequence.

The classification is parallelized through RcppParallel employing Intel TBB and TinyThread. By default all available processing cores are used. This can be changed using the function setParallel.

Value

A character vector with the predicted taxa, one for each sequence.

Author(s)

Kristian Hovde Liland and Lars Snipen.

References

Wang, Q, Garrity, GM, Tiedje, JM, Cole, JR (2007). Naive Bayesian Classifier for Rapid Assignment of rRNA Sequences into the New Bacterial Taxonomy. Applied and Enviromental Microbiology, 73: 5261-5267.

See Also

rdpTrain.

Examples

```
data("small.16S")
seq <- small.16S$Sequence
tax <- sapply(strsplit(small.16S$Header,split=" "),function(x){x[2]})
## Not run:
trn <- rdpTrain(seq,tax)
primer.515f <- "GTGYCAGCMGCCGCGGTAA"
primer.806rB <- "GGACTACNVGGGTWTCTAAT"
reads <- amplicon(seq, primer.515f, primer.806rB)
predicted <- rdpClassify(unlist(reads[nchar(reads)>0]),trn)
print(predicted)

## End(Not run)
```

---

rdpTrain *Training the RDP classifier*

Description

Training the RDP presence/absence K-mer method on sequence data.

Usage

```
rdpTrain(sequence, taxon, K = 8, cnames = FALSE)
```

Arguments

sequence	Character vector of 16S sequences.
taxon	Character vector of taxon labels for each sequence.
K	Word length (integer).
cnames	Logical indicating if column names should be added to the trained model matrix.

Details

The training step of the RDP method means looking for K-mers on all sequences, and computing the probability of each K-mer being present for each unique taxon. This is an attempt to re-implement the method described by Wang et tal (2007), but without the bootstrapping. See that publications for all details.

The word-length `K` is by default 8, since this is the value used by Wang et al. Larger values may lead to memory-problems since the trained model is a matrix with 4^K columns. Adding the K-mers as column names will slow down all computations.

The relative taxon sizes are also computed, and returned as an attribute to the model matrix. They may be used as empirical priors in the classification step.

Value

A list with two elements. The first element is `Method`, which is the text `"RDPclassifier"` in this case. The second element is `Fitted`, which is a matrix with one row for each unique `taxon` and one column for each possible word of length `K`. The value in row i and column j is the probability that word j is present in taxon i.

Author(s)

Kristian Hovde Liland and Lars Snipen.

References

Wang, Q, Garrity, GM, Tiedje, JM, Cole, JR (2007). Naive Bayesian Classifier for Rapid Assignment of rRNA Sequences into the New Bacterial Taxonomy. Applied and Enviromental Microbiology, 73: 5261-5267.

See Also

`rdpClassify`.

Examples

```
# See examples for rdpClassify.
```

`read_names_dmp` *Read and write the names.dmp*

Description

Reads and writes the file names.dmp of the NCBI Taxonomy

Usage

```
read_names_dmp(filename)
write_names_dmp(names.dmp, filename)
```

Arguments

filename	name of file to be read or written to (text).
names.dmp	a names.dmp table (see details below).

Details

The file pair names.dmp and nodes.dmp describe a taxonomy tree. The read_names_dmp reads a file formatted as the names.dmp file from the NCBI Taxonomy database (https://www.ncbi.nlm.nih.gov/taxonomy/). This is represented as a tibble in R.

The write_names_dmp will write a table with the proper columns (see below) to a file, adding the separators of the NCBI format.

Value

The read_names_dmp returns a tibble with the columns: tax_id (integers), name_txt (text), unique_name (text) and name_class (text).

Author(s)

Lars Snipen.

See Also

read_nodes_dmp.

---

read_nodes_dmp *Read and write the nodes.dmp*

Description

Reads and writes the file nodes.dmp of the NCBI Taxonomy

Usage

```
read_nodes_dmp(filename)
write_nodes_dmp(names.dmp, filename)
```

Arguments

filename	name of file to be read or written to.
nodes.dmp	a nodes.dmp table (see details below).

Details

The file pair `names.dmp` and `nodes.dmp` describe a taxonomy tree. The `read_nodes_dmp` reads a file formatted as the `nodes.dmp` file from the NCBI Taxonomy database (https://www.ncbi.nlm.nih.gov/taxonomy/). This is represented as a `tibble` in R.

The `write_nodes_dmp` will write a table with the proper columns (see below) to a file, adding the separators of the NCBI format.

The `nodes.dmp` table downloaded from NCBI will contain many columns, but only the first 3 of them are relevant for parsing the taxonomy tree. Only these first three columns are read and used by these functions, additional columns are ignored.

Value

The `read_nodes_dmp` returns a `tibble` with the columns: `tax_id` (integers), `parent_tax_id` (integers) and `rank` (text).

Author(s)

Lars Snipen.

See Also

`read_nodes_dmp`.

---

setParallel *Set number of parallel threads*

Description

Simple function to set the number of threads to use in parallel computations. The default equals all available logical cores. An integer is interpreted as the number of threads. A numeric < 1 is interpreted as a proportion of the avialable logical cores.

Usage

```
setParallel(C = NULL)
```

Arguments

`C`	a scalar indicating the number of threads, default = NULL (#available logical cores)

Value

NULL, returned silently.

Examples

```
## Not run:
setParallel() # Use all available logical cores.

## End(Not run)
```

`small.16S` *A small example data set*

Description

A `tibble` object (`data.frame`) with some 16S sequences with taxon information.

Usage

```
data(small.16S)
```

Details

This is a `tibble` object (`data.frame`) with 71 sequences used in some examples. The taxonomy information for each sequence follows the ConTax format, see the `microcontax` package for more details.

Author(s)

Hilde Vinje, Kristian Hovde Liland, Lars Snipen.

Examples

```
data(small.16S)
str(small.16S)
```

`subset_clade` *Subset the taxonomy tree from the root*

Description

Retrieves a clade from the taxonomy tree.

Usage

```
subset_clade(root.tax.id, nodes.dmp)
```

Arguments

root.tax.id	the tax_id of the clade root (integer).
nodes.dmp	a nodes.dmp table.

Details

This function retrieves a clade from the taxonomy tree listed in the table nodes.dmp (see read_nodes_dmp). The clade starts at the single taxon specified in root_tax_id and contains all descending branches.

If you want to subset the taxonomy tree based on leaf nodes, see subset_tree.

Value

A subset of the table nodes.dmp containing the clade that descends from root.tax.id.

Author(s)

Lars Snipen.

See Also

read_nodes_dmp, subset_tree.

---

subset_tree *Subset the taxonomy tree from leaves*

Description

Retrieves a sub-tree from the taxonomy tree.

Usage

```
subset_tree(leaf_tax_id, nodes.dmp)
```

Arguments

leaf.tax.id	a vector of tax_id (integers) of the branch leaf/leaves.
nodes.dmp	a nodes.dmp table.

Details

This function retrieves a sub-tree by starting at the vector of specified leaf_tax_id and collect all branches ending at these nodes. Thus, it will in general not be a clade. If you want to subset an entire clade, see subset_clade.

Value

A subset of the table nodes.dmp containing the tax_id's of the branches ending at leaf_tax_id.

Author(s)

Lars Snipen.

See Also

read_nodes_dmp, subset_clade.

---

taxMachine *Classifying 16S sequences*

Description

Multinomial classification of 16S sequence data.

Usage

```
taxMachine(
  sequence,
  model.in.memory = TRUE,
  model.on.disk = FALSE,
  verbose = TRUE,
  chunk.size = 10000
)
```

Arguments

sequence	Character vector with DNA sequences.
model.in.memory	Logical indicating if model should be cached in memory (default=TRUE).
model.on.disk	Logical or text, for reading/saving models, see Deatils below (default=FALSE).
verbose	Logical, if TRUE progress is reported during computations (default=TRUE).
chunk.size	The number of sequence to classify in each iteration of the loop (default=10000).

Details

This function provides an optimized taxonomy classifications from 16S sequence data.

All sequences are classified to the genus level based on a Multinomial model (see multinomTrain) trained on the designed consensus taxonomy data set contax.trim found in the R-package microcontax. The word length K=8 has been used in the model.

To avoid saving fitted models in the package, a model is trained the first time you run taxMachine in an R session. This takes only a few seconds, and the result is cached for latter use if model.in.memory is TRUE.

If a path to an existing file with a trained model is supplied in model.on.disk, this Multinomial model is read from the file and used. If a path to a new file is supplied, the trained Multinomial model will be saved to that file. The default (model.on.disk=FALSE), means no files are read/saved,

while model.on.disk=TRUE will attempt to load/save models from the microclass/extdata directory.

Both verbose and chunk.size are used to monitor the progress, which is nice when classifying huge data sets, since this will take some time.

Value

A data.frame with one row for each sequence. The columns are Genus, D.score, R.score and P.recognize.

Genus is the predicted genus for each sequence. Note that all sequences get a prediction, but may still be more or less reliable.

The D.score is a measure of how the predicted genus wins over all other genera in the race for being the chosen one. A large D.score means the winner stands out clearly, and we can be confident it is the correct genus. A D.score close to 0 means we have an uncertain classification. Only D.scores below 1.0, should be of any concern, see Liland et al (2016) for details.

The R.score is a measure of the models ability to recognize the sequence. The more negative the R.score gets, the more unusual the sequence is compared to the training set (the contax.trim data set). The P.recognize is a rough probability of seing an R.score this small, or smaller, given the training data. Thus, a very small P.recognize means the sequence is not really recognized, and the classification is worthless. A very negative R.score indicates either not 16S at all, many sequencing errors that has destroyed the read, or a completely new taxon never seen before. See Liland et al (2016) for details.

Author(s)

Lars Snipen and Kristian Hovde Liland

References

Liland, KH, Vinje, H, Snipen, L (2016). microclass - An R-package for 16S taxonomy classification. BMC Bioinformatics, xx:yy.

See Also

KmerCount, multinomClassify.

Examples

```
## Not run:
data(small.16S)
tax.tab <- taxMachine(small.16S$Sequence)

## End(Not run)
```

Index

∗ **package**
    microclass-package, 2

as_tibble, *8*

blastClassify16S, 3, *4, 5*
blastDbase16S, *3, 4*, 4
bracken_read_report, 5, *14*
branch_list2qiime, 6
branch_list2table, 7
branch_name2taxid *(*branch_taxid2name*)*, 9
branch_prune, *7*, 8
branch_retrieve, 8, *10*
branch_table2list *(*branch_list2table*)*, 7
branch_taxid2name, *6*, *9*, 9
branch_taxid2name, *(*branch_taxid2name*)*, 9

clr, 10
contax.trim, *12*, *25, 26*

getClass *(*getDomain*)*, 11
getDomain, 11
getFamily *(*getDomain*)*, 11
getGenus *(*getDomain*)*, 11
getOrder *(*getDomain*)*, 11
getPhylum *(*getDomain*)*, 11
getTag *(*getDomain*)*, 11
getTaxonomy *(*getDomain*)*, 11

KmerCount, 12, *16*, *18*, *26*
kraken2_read_report, *5, 6*, 13, *14, 15*
kraken2_read_table, *14*, 14

match, *10*
medoids, *12*
microclass *(*microclass-package*)*, 2
microclass-package, 2
microcontax, *23*
multinomClassify, *13*, 15, *18*, *26*
multinomTrain, *13*, *15, 16*, 17, *25*

rdpClassify, 18, *20*
rdpTrain, *18, 19*, 19
read_names_dmp, *10*, 20
read_nodes_dmp, *9*, *21*, 21, *22*, *24, 25*

setParallel, *16*, *18*, 22
small.16S, 23
subset_clade, *9*, 23, *24, 25*
subset_tree, *24*, 24

taxMachine, 25

write_names_dmp *(*read_names_dmp*)*, 20
write_nodes_dmp *(*read_nodes_dmp*)*, 21