Package ‘insect’

May 8, 2026

Type Package

Title Informatic Sequence Classification Trees

Version 1.4.4

Author Shaun Wilkinson [aut, cre]

Maintainer Shaun Wilkinson <shaunpwilkinson@gmail.com>

Description Provides tools for probabilistic taxon assignment with informatic sequence classification trees. See Wilkinson et al (2018) <doi:10.7287/peerj.preprints.26812v1>.

License GPL-3

LazyData TRUE

URL https://github.com/shaunpwilkinson/insect

BugReports https://github.com/shaunpwilkinson/insect/issues

Encoding UTF-8

Imports ape (>= 3.0.0), aphid (>= 1.3.1), kmer (>= 1.1.0), openssl, phylogram (>= 2.0.0), RANN, seqinr, stats, utils, xml2

Suggests knitr, rmarkdown, testthat

VignetteBuilder knitr

RoxygenNote 7.3.1

NeedsCompilation no

Repository CRAN

Date/Publication 2025-05-24 08:10:02 UTC

Contents

allocateCVI 2
classify 3
conversion 7
demultiplex 8
disambiguate 9
encoding 10

expand . . . 11
get_lineage . . . 13
get_taxID . . . 14
hash . . . 15
insect . . . 16
join . . . 17
learn . . . 17
manipulate . . . 21
prune_taxonomy . . . 23
purge . . . 24
qfilter . . . 25
rc . . . 26
read . . . 27
replicate . . . 29
samoa . . . 30
searchGB . . . 30
shave . . . 32
stitch . . . 33
taxonomy . . . 34
trim . . . 35
virtualFISH . . . 36
virtualPCR . . . 38
whales . . . 40
whale_taxonomy . . . 41
write . . . 42

Index **44**

`allocateCVI` *Allocate sequences for cross validation by identity.*

Description

This function takes a reference sequence database and allocates each sequence to either a query set (a.k.a. test set) or a training set, in order to cross validate a supervised taxon classifier. The method is based on that of Edgar (2018), but uses recursive divisive clustering and retains all sequences rather than discarding those that violate the top-hit identity constraint.

Usage

```
allocateCVI(x, threshold = 0.9, allocate = "max", ...)
```

Arguments

`x`	a set of reference sequences. Can be a "DNAbin" object or a named vector of upper-case DNA character strings.
`threshold`	numeric between 0 and 1 giving the identity threshold for sequence allocation.
`allocate`	character giving the method to use to allocate eligible sequences to the query set. Options are "max" (default) which chooses the largest node from each pair in order to maximize the size of the query set, or "sample", which randomly chooses one node from each eligible pair.
`...`	further arguments to pass to "kmeans"

Value

a logical vector the same length as the input object, indicating which sequences should be allocated to the query set

Author(s)

Shaun Wilkinson

References

Edgar RC (2018) Accuracy of taxonomy prediction for 16S rRNA and fungal ITS sequences. PeerJ 6:e4652. DOI 10.7717/peerj.4652

Examples

```
data(whales)
allocateCVI(whales)
```

`classify` *Tree-based sequence classification.*

Description

`"classify"` assigns taxon IDs to DNA sequences using an informatic sequence classification tree.

Usage

```
classify(
  x,
  tree,
  threshold = 0.8,
  decay = FALSE,
  ping = 0.98,
  mincount = 5,
  offset = 0,
```

```
  ranks = c("kingdom", "phylum", "class", "order", "family", "genus", "species"),
  species = "ping100",
  tabulize = FALSE,
  metadata = FALSE,
  cores = 1
)
```

Arguments

x	a sequence or set of sequences. Can be a "DNAbin" or "AAbin" object or a named vector of upper-case DNA character strings.
tree	an object of class "insect" (see learn for details).
threshold	numeric between 0 and 1 giving the minimum Akaike weight for the recursive classification procedure to continue toward the leaves of the tree. Defaults to 0.8.
decay	logical indicating whether the decision to terminate the classification process should be made based on decaying Akaike weights (at each node, the Akaike weight of the selected model is multiplied by the Akaike weight of the selected model at the parent node) or whether each Akaike weight should be calculated independently of that of the parent node. Defaults to FALSE (the latter).
ping	logical or numeric (between 0 and 1) indicating whether a nearest neighbor search should be carried out, and if so, what the minimum distance to the nearest neighbor should be for the the recursive classification algorithm to be skipped. If TRUE and the query sequence is identical to at least one of the training sequences used to learn the tree, the common ancestor of the matching training sequences is returned with an score of NA. If a value between 0 and 1 is provided, the common ancestor of the training sequences with similarity greater than or equal to 'ping' is returned, again with a score of NA. If ping is set to 0 or FALSE, the recursive classification algorithm is applied to all sequences, regardless of proximity to those in the training set. For high values (e.g. ping >= 0.98) the output will generally specify the taxonomic ID to species or genus level; however a higher rank may be returned for low-resolution genetic markers.
mincount	integer, the minimum number of training sequences belonging to a selected child node for the classification to progress. Defaults to 5.
offset	log-odds score offset parameter governing whether the minimum score is met at each node. Defaults to 0. Values above 0 increase precision (fewer type I errors), values below 0 increase recall (fewer type II errors).
ranks	character vector giving the taxonomic ranks to be included in the output table. Must be a valid rank from the taxonomy database attributed to the classification tree (attr(tree, "taxonomy")). Set to NULL to exclude taxonomic ranks from the output table.
species	character string, indicating whether to include all species-level classifications in the output (species = 'all'), only those generated by exact matching ("ping100"; the default setting), only those generated by exact matching or near-neighbor searching (species = 'ping'). If species = "ping" or species = "ping100",

non-matched species are returned at genus level. Alternatively, if species = 'none', all species-level classifications are returned at genus level.

`tabulize` logical indicating whether sequence counts should be attached to the output table. If TRUE, the output table will have one row for each unique sequence, and columns will include counts for each sample (where samples names precede sequence identifiers in the input object; see details below).

`metadata` logical indicating whether to include additional columns containing the paths, individual node scores and reasons for termination. Defaults to FALSE. Included for advanced use and debugging.

`cores` integer giving the number of processors for multithreading (defaults to 1). This argument may alternatively be a 'cluster' object, in which case it is the user's responsibility to close the socket connection at the conclusion of the operation, for example by running `parallel::stopCluster(cores)`. The string 'autodetect' is also accepted, in which case the maximum number of cores to use is one less than the total number of cores available.

Details

This function requires a pre-computed classification tree of class "insect", which is a dendrogram object with additional attributes (see `learn` for details). Query sequences obtained from the same primer set used to construct the tree are classified to produce taxonomic IDs with an associated degree of confidence. The classification algorithm works as follows: starting from the root node of the tree, the log-likelihood of the query sequence (the log-probability of the sequence given a particular model) is computed for each of the models occupying the two child nodes using the forward algorithm (see Durbin et al. (1998)). The competing likelihood values are then compared by computing their Akaike weights (Johnson and Omland, 2004). If one model is overwhelmingly more likely to have produced the sequence than the other, that child node is chosen and the classification is updated to reflect the taxonomic ID stored at the node. This classification procedure is repeated, continuing down the tree until either an inconclusive result is returned by a model comparison test (i.e. the Akaike weight is lower than a pre-defined threshold, e.g. 0.9), or a terminal leaf node is reached, at which point a species-level classification is generally returned. The function outputs a table with one row for each input sequence Output table fields include "name" (the unique sequence identifier), "taxID" (the taxonomic identification number from the taxonomy database), "taxon" (the name of the taxon), "rank" (the rank of the taxon, e.g. species, genus family, etc), and "score" (the Akaike weight from the model selection procedure). Note that the default behavior is for the Akaike weight to 'decay' as it moves down the tree, by computing the cumulative product of all preceding Akaike weight values. This minimizes the chance of type I taxon ID errors (overclassifications and misclassifications). The output table also includes the higher taxonomic ranks specified in the `ranks` argument, and if `metadata = TRUE` additional columns are included called "path" (the path of the sequence through the classification tree), "scores" (the scores at each node through the tree, UTF-8-encoded), and "reason" outlining why the recursive classification procedure was terminated:

- 0 reached leaf node
- 1 failed to meet minimum score threshold at inner node
- 2 failed to meet minimum score of training sequences at inner node
- 3 sequence length shorter than minimum length of training sequences at inner node
- 4 sequence length exceeded maximum length of training sequences at inner node

- 5 nearest neighbor in training set does not belong to selected node (obsolete)
- 6 node is supported by too few sequences
- 7 reserved
- 8 sequence could not be translated (amino acids only)
- 9 translated sequence contains stop codon(s) (amino acids only)

Additional columns detailing the nearest neighbor search include "NNtaxID", "NNtaxon", "NNrank", and "NNdistance".

Value

a data.frame.

Author(s)

Shaun Wilkinson

References

Durbin R, Eddy SR, Krogh A, Mitchison G (1998) Biological sequence analysis: probabilistic models of proteins and nucleic acids. Cambridge University Press, Cambridge, United Kingdom.

Johnson JB, Omland KS (2004) Model selection in ecology and evolution. *Trends in Ecology and Evolution*. **19**, 101-108.

See Also

learn

Examples

```
data(whales)
data(whale_taxonomy)
## use all sequences except first one to train the classifier
set.seed(999)
tree <- learn(whales[-1], db = whale_taxonomy, maxiter = 5, cores = 2)
## find predicted lineage for first sequence
classify(whales[1], tree)
## compare with actual lineage
taxID <- as.integer(gsub(".+\\|", "", names(whales)[1]))
get_lineage(taxID, whale_taxonomy)
```

`conversion` *Convert sequences between binary and character string formats.*

Description

These functions convert DNA and amino acid sequences in "DNAbin" or "AAbin" format to concatenated character strings, and vice versa.

Usage

```
dna2char(x)

aa2char(x)

char2dna(z, simplify = FALSE)

char2aa(z, simplify = FALSE)
```

Arguments

`x`	a "DNAbin" or "AAbin" object.
`z`	a vector of concatenated strings representing DNA or amino acid sequences in upper case.
`simplify`	logical indicating whether length-one "DNAbin" or "AAbin" objects should be simplified to vectors. Defaults to FALSE.

Details

These functions are used to convert concatenated character strings (e.g. "TAACGC") to binary format and vice versa. To convert DNAbin and AAbin objects to non-concatenated character objects (e.g. `c("T", "A", "A", "C", "G", "C")`) refer to the `ape` package functions `as.character.DNAbin` and `as.character.AAbin`. Likewise the `ape` package functions `as.DNAbin` and `as.AAbin` are used to convert non-concatenated character objects to binary format.

Value

`dna2char` and `aa2char` return vectors of upper case character strings. `char2dna` and `char2aa` return "DNAbin" and "AAbin" objects, respectively. These will be lists unless the input object has length one and simplify = TRUE, in which case the returned object will be a vector.

Author(s)

Shaun Wilkinson

References

Paradis E, Claude J, Strimmer K, (2004) APE: analyses of phylogenetics and evolution in R language. *Bioinformatics* **20**, 289-290.

Paradis E (2007) A bit-level coding scheme for nucleotides. https://emmanuelparadis.github.io/misc/BitLevelCodingScheme_20April2007.pdf.

Paradis E (2012) Analysis of Phylogenetics and Evolution with R (Second Edition). Springer, New York.

Examples

```
char2dna("TAACGC")
char2aa("VGAHAGEY")
dna2char(char2dna("TAACGC"))
aa2char(char2aa("VGAHAGEY"))
char2dna(list(seq1 = "TAACGC", seq2 = "ATTGCG"))
char2aa(list(seq1 = "VGAHAGEY", seq2 = "VNVDEV"))
```

---

demultiplex *Demultiplex merged FASTQ*

Description

This function is used to demultiplex FASTQ files containing sequence reads with index and primer sequences still attached. The function trims the tags and primers, and exports two FASTQ files for each forward-reverse index combination.

Usage

```
demultiplex(
  R1,
  R2 = NULL,
  tags,
  up,
  down,
  destdir = "demux",
  adapter1 = NULL,
  adapter2 = NULL
)
```

Arguments

R1	character string giving the path to the first FASTQ file
R2	character string giving the path to the second FASTQ file. Set to NULL for single-end reads.

`tags`	named character vector specifying the unique forward:reverse tag combinations used in the run. Each tag combination should be entered as an upper case character string delimited by a colon (e.g. "ATCGACAC:ATGCACTG") and named according to the unique sample ID.
`up, down`	upper case character strings giving the forward and reverse primer sequences.
`destdir`	character string giving the path to the directory where the new FASTQ files should be written.
`adapter1`	the forward flowcell adapter sequence to check and trim (set NULL to ignore). For standard Illumina MiSeq forward adapter set to "AATGATACGGCGACCACCGAGATCTACAC" (paired end sequencing only).
`adapter2`	the reverse flowcell adapter sequence to check and trim (set NULL to ignore). For standard Illumina MiSeq reverse adapter set to "CAAGCAGAAGACGGCATACGAGAT" (single or paired end sequencing).

Value

NULL (invisibly)

Author(s)

Shaun Wilkinson

`disambiguate` *Convert oligonucleotide sequences into regular expressions.*

Description

This function is used to convert an oligonucleotide sequence into a regular expression that can be used to query a sequence dataset. This is generally used for finding and removing primer, adapter and/or index sequences.

Usage

```
disambiguate(z)
```

Arguments

`z`	a concatenated string representing a DNA oligonucleotide sequence, possibly with IUPAC ambiguity codes (all in upper case).

Value

a regular expression.

Author(s)

Shaun Wilkinson

Examples

```
disambiguate("GGWACWGGWTGAACWGTWTAYCCYCC")
```

---

encoding *Encode and decode profile HMMs in raw byte format.*

Description

These functions are used to compress and decompress profile hidden Markov models for DNA to improve memory efficiency.

Usage

```
encodePHMM(x)

decodePHMM(z)
```

Arguments

x	an object of class "PHMM"
z	a raw vector in the encodePHMM schema.

Details

Profile HMMs used in tree-based classification usually include many parameters, and hence large trees with many PHMMs can occupy a lot of memory. Hence a basic encoding system was devised to store the emission and transition probabilities in raw-byte format to three (nearly four) decimal places. This does not seem to significantly affect the accuracy of likelihood scoring, and has a moderate impact on classification speed, but can reduce the memory allocation requirements for large trees by up to 95 percent.

Value

encodePHMM returns a raw vector. `decodePHMM` returns an object of class "PHMM" (see Durbin et al (1998) and the `aphid` package for more details on profile hidden Markov models).

Author(s)

Shaun Wilkinson

References

Durbin R, Eddy SR, Krogh A, Mitchison G (1998) Biological sequence analysis: probabilistic models of proteins and nucleic acids. Cambridge University Press, Cambridge, United Kingdom.

Examples

```
## generate a simple classification tree with two child nodes
data(whales)
data(whale_taxonomy)
tree <- learn(whales, db = whale_taxonomy, recursive = FALSE)
## extract the omnibus profile HMM from the root node
PHMM0 <- decodePHMM(attr(tree, "model"))
## extract the profile HMM from the first child node
PHMM1 <- decodePHMM(attr(tree[[1]], "model"))
```

---

expand *Expand an existing classification tree.*

Description

This function is used to grow an existing classification tree, typically using more relaxed parameter settings than those used when the tree was created, or if fine-scale control over the tree-learning operation is required.

Usage

```
expand(
  tree,
  clades = "0",
  refine = "Viterbi",
  iterations = 50,
  nstart = 20,
  minK = 2,
  maxK = 2,
  minscore = 0.9,
  probs = 0.5,
  retry = TRUE,
  resize = TRUE,
  maxsize = 1000,
  recursive = TRUE,
  cores = 1,
  quiet = FALSE,
  verbose = FALSE,
  ...
)
```

Arguments

`tree`	an object of class `"insect"`.
`clades`	a vector of character strings giving the binary indices matching the labels of the nodes that are to be expanded. Defaults to "0", meaning all subclades are expanded. See below for further details on clade indexing.

`refine` character string giving the iterative model refinement method to be used in the partitioning process. Valid options are `"Viterbi"` (Viterbi training; the default option) and `"BaumWelch"` (a modified version of the Expectation-Maximization algorithm).

`iterations` integer giving the maximum number of training-classification iterations to be used in the splitting process. Note that this is not necessarily the same as the number of Viterbi training or Baum Welch iterations to be used in model training, which can be set using the argument `"maxiter"` (eventually passed to `train` via the dots argument "...").

`nstart` integer. The number of random starting sets to be chosen for initial k-means assignment of sequences to groups. Defaults to 20.

`minK` integer. The minimum number of furications allowed at each inner node of the tree. Defaults to 2 (all inner nodes are bifuricating).

`maxK` integer. The maximum number of furications allowed at each inner node of the tree. Defaults to 2 (all inner nodes are bifuricating).

`minscore` numeric between 0 and 1. The minimum acceptable value for the *n*th percentile of Akaike weights (where *n* is the value given in `"probs"`, for the node to be split and the recursion process to continue. At any given node, if the *n*th percentile of Akaike weights falls below this threshold, the recursion process for the node will terminate. As an example, if `minscore = 0.95` and `probs = 0.5` (the default settings), and after generating two candidate PHMMs to occupy the candidate subnodes the median Akaike weight is less than 0.95, the splitting process will terminate and the function will simply return the unsplit root node.

`probs` numeric between 0 and 1. The percentile of Akaike weights to test against the minimum score threshold given in `"minscore"`.

`retry` logical indicating whether failure to split a node based on the criteria outlined in 'minscore' and 'probs' should prompt a second attempt with different initial groupings. These groupings are based on maximum kmer frequencies rather than k-means division, which can give suboptimal groupings when the cluster sizes are different (due to the up-weighting of larger clusters in the k-means algorithm).

`resize` logical indicating whether the models should be free to change size during the training process or if the number of modules should be fixed. Defaults to TRUE. Only applicable if `refine = "Viterbi"`.

`maxsize` integer giving the upper bound on the number of modules in the PHMMs. If NULL, no maximum size is enforced.

`recursive` logical indicating whether the splitting process should continue recursively until the discrimination criteria are not met (TRUE; default), or whether a single split should take place at each of the nodes specified in `clades`.

`cores` integer giving the number processors for multithreading. Defaults to 1. This argument may alternatively be a 'cluster' object, in which case it is the user's responsibility to close the socket connection at the conclusion of the operation, e.g. by running `parallel::stopCluster(cores)`. The string 'autodetect' is also accepted, in which case the maximum number of cores to use is one less than the total number of cores available.

quiet	logical indicating whether feedback should be printed to the console.
verbose	logical indicating whether extra feedback should be printed to the console, including progress at each split.
...	further arguments to be passed on to train).

Details

The clade indexing system used here is based on character strings, where "0" refers to the root node, "01" is the first child node, "02" is the second child node, "011" is the first child node of the first child node, etc. Note that this means each node cannot have more than 9 child nodes.

Value

an object of class "insect".

Author(s)

Shaun Wilkinson

See Also

learn.

Examples

```
data(whales)
data(whale_taxonomy)
## split the first node
set.seed(123)
tree <- learn(whales, db = whale_taxonomy, recursive = FALSE)
## expand only the first clade
tree <- expand(tree, clades = "1")
```

---

get_lineage *Get full lineage details from a taxonomic ID number.*

Description

This function derives the full lineage of a taxon ID number from a given taxonomy database.

Usage

```
get_lineage(taxIDs, db, simplify = TRUE, numbers = FALSE, cores = 1)
```

Arguments

`taxIDs`	integer or vector of integers giving the taxonomic ID number(s).
`db`	a taxonomy database (a data.frame object). See `taxonomy` for details.
`simplify`	logical indicating whether a single lineage derived from a length-one input should be simplified from a list to a named character vector. Defaults to TRUE.
`numbers`	logical indicating whether the output string(s) should be comprised of the taxonomic ID numbers rather than taxon names. Defaults to FALSE.
`cores`	integer giving the number of processors for multithreading (Defaults to 1). This argument may alternatively be a 'cluster' object, in which case it is the user's responsibility to close the socket connection at the conclusion of the operation, for example by running `parallel::stopCluster(cores)`. The string 'autodetect' is also accepted, in which case the maximum number of cores to use is one less than the total number of cores available. Note that in this case there may be a tradeoff in terms of speed depending on the number and size of sequences to be processed, due to the extra time required to initialize the cluster.

Value

the full lineage as a named character vector, or list of named character vectors if the length of the input object is > 1 or simplify = FALSE. "names" attributes are taxonomic ranks.

Author(s)

Shaun Wilkinson

References

Federhen S (2012) The NCBI Taxonomy database. *Nucleic Acids Research* **40**, D136-D143. doi:10.1093/nar/gkr1178.

https://www.ncbi.nlm.nih.gov/taxonomy/

Examples

```
data(whales)
data(whale_taxonomy)
taxIDs <- as.integer(gsub(".+\\|", "", names(whales)[1:2]))
get_lineage(taxIDs, db = whale_taxonomy)
```

`get_taxID` *Get taxon ID from taxonomy database.*

Description

This function returns the unique ID for a specified taxon name by looking up a taxonomy database.

Usage

```
get_taxID(lineage, db, multimatch = NA)
```

Arguments

`lineage`	character vector of taxon names or semicolon-delimited lineage strings.
`db`	a valid taxonomy database (as a data.frame object). See `taxonomy` for details.
`multimatch`	integer giving the value to return if the query matches multiple entries. Defaults to NA_integer_.

Value

An integer giving the unique taxon ID, or NA if the taxon is not found in the database.

Author(s)

Shaun Wilkinson

References

Federhen S (2012) The NCBI Taxonomy database. *Nucleic Acids Research* **40**, D136-D143. doi:10.1093/nar/gkr1178.

https://www.ncbi.nlm.nih.gov/taxonomy/

Examples

```
data(whale_taxonomy)
get_taxID("Odontoceti", db = whale_taxonomy)
```

---

hash — *Convert sequences to MD5 hashes.*

Description

This function converts DNA or amino acid sequences to 128-bit MD5 hash values for efficient duplicate identification and dereplication.

Usage

```
hash(x)
```

Arguments

`x`	a sequence or list of sequences, either in character string, character vector, or raw byte format (eg DNAbin or AAbin objects).

Details

This function uses the `md5` function from the openSSL library (https://www.openssl.org/) to digest sequences to 128-bit hashes. These can be compared using base functions such as `duplicated` and `unique`, for fast identification and management of duplicate sequences in large datasets.

Value

a character vector.

Author(s)

Shaun Wilkinson

References

Ooms J (2017) openssl: toolkit for encryption, signatures and certificates based on OpenSSL. R package version 0.9.7. https://CRAN.R-project.org/package=openssl

Examples

```
data(whales)
hashes <- hash(whales)
sum(duplicated(hashes))
```

---

insect — *Informatic sequence classification trees.*

Description

R package 'insect' contains functions to create and use classification trees for DNA meta-barcoding. This enables users to quickly and accurately assign the taxonomic identities of large numbers of DNA barcodes or other informative sequences generated on one of the standard high-throughput sequencing platforms.

Author(s)

Maintainer: Shaun Wilkinson <shaunpwilkinson@gmail.com>

See Also

Useful links:

- https://github.com/shaunpwilkinson/insect
- Report bugs at https://github.com/shaunpwilkinson/insect/issues

join — *Concatenate DNAbin objects while preserving attributes.*

Description

This function joins two or more DNAbin objects, retaining any attributes whose lengths match those of the input objects (e.g. "species", "lineage" and/or "taxID" attributes).

Usage

```
join(...)
```

Arguments

...	DNAbin objects to be concatenated.

Value

an object of class DNAbin.

Author(s)

Shaun Wilkinson

See Also

subset.DNAbin.

Examples

```
data(whales)
whales1 <- whales[1:10]
whales2 <- whales[11:19]
join(whales1, whales2)
```

learn — *Informatic sequence classification tree learning.*

Description

This function learns a classification tree from a reference sequence database using a recursive partitioning procedure.

Usage

```
learn(
  x,
  db = NULL,
  model = NULL,
  refine = "Viterbi",
  iterations = 50,
  nstart = 20,
  minK = 2,
  maxK = 2,
  minscore = 0.95,
  probs = 0.5,
  retry = FALSE,
  resize = TRUE,
  maxsize = 1000,
  recursive = TRUE,
  cores = 1,
  quiet = FALSE,
  verbose = FALSE,
  numcode = NULL,
  frame = NULL,
  ...
)
```

Arguments

`x` a reference database of class`"DNAbin"` representing a list of DNA sequences to be used as the training data. All sequences should be from the same genetic region of interest and be globally alignable (i.e. without unjustified end-gaps). The sequences must have "names" attributes, either in RDP format (containing semicolon-delimited lineage strings), or that include taxonomic ID numbers corresponding with those in the taxonomy database db (separated from the sequence ID by a "|" character). For example: "AF296347|30962", "AF296346|8022", "AF296345|8017", etc. See `searchGB` for more details on creating the reference sequence database and `taxonomy` for the associated heirarchical taxonomic database.

`db` a heirarchical taxonomy database in the form of a data.frame. Cannot be NULL unless training data is in RDP format (containing semicolon delimited lineage strings). The object should have four columns, labeled "taxID", "parent_taxID", "rank" and "name". The first two should be numeric, and all ID numbers in the "parent_taxID" column should link to those in the "taxID" column. This excludes the first row, which should have `parent_taxID = 0` and `name = "root"`. See `taxonomy` for more details.

`model` an optional object of class `"PHMM"` providing the starting parameters. Used to train (optimize parameters for) subsequent nested models to be positioned at successive sub-nodes. If NULL, the root model is derived from the sequence list prior to the recursive partitioning process.

`refine` character string giving the iterative model refinement method to be used in the partitioning process. Valid options are `"Viterbi"` (Viterbi training; the default option) and `"BaumWelch"` (a modified version of the Expectation-Maximization algorithm).

`iterations` integer giving the maximum number of training-classification iterations to be used in the splitting process. Note that this is not necessarily the same as the number of Viterbi training or Baum Welch iterations to be used in model training, which can be set using the argument `"maxiter"` (eventually passed to `train` via the dots argument "...").

`nstart` integer. The number of random starting sets to be chosen for initial k-means assignment of sequences to groups. Defaults to 20.

`minK` integer. The minimum number of furications allowed at each inner node of the tree. Defaults to 2 (all inner nodes are bifuricating).

`maxK` integer. The maximum number of furications allowed at each inner node of the tree. Defaults to 2 (all inner nodes are bifuricating).

`minscore` numeric between 0 and 1. The minimum acceptable value for the *n*th percentile of Akaike weights (where *n* is the value given in `"probs"`, for the node to be split and the recursion process to continue. At any given node, if the *n*th percentile of Akaike weights falls below this threshold, the recursion process for the node will terminate. As an example, if `minscore = 0.95` and `probs = 0.5` (the default settings), and after generating two candidate PHMMs to occupy the candidate subnodes the median Akaike weight is less than 0.95, the splitting process will terminate and the function will simply return the unsplit root node.

`probs` numeric between 0 and 1. The percentile of Akaike weights to test against the minimum score threshold given in `"minscore"`.

`retry` logical indicating whether failure to split a node based on the criteria outlined in 'minscore' and 'probs' should prompt a second attempt with different initial groupings. These groupings are based on maximum kmer frequencies rather than k-means division, which can give suboptimal groupings when the cluster sizes are different (due to the up-weighting of larger clusters in the k-means algorithm).

`resize` logical indicating whether the models should be free to change size during the training process or if the number of modules should be fixed. Defaults to TRUE. Only applicable if `refine = "Viterbi"`.

`maxsize` integer giving the upper bound on the number of modules in the PHMMs. If NULL, no maximum size is enforced.

`recursive` logical indicating whether the splitting process should continue recursively until the discrimination criteria are not met (TRUE; default), or whether a single split should take place at the root node.

`cores` integer giving the number processors for multithreading. Defaults to 1. This argument may alternatively be a 'cluster' object, in which case it is the user's responsibility to close the socket connection at the conclusion of the operation, e.g. by running `parallel::stopCluster(cores)`. The string 'autodetect' is also accepted, in which case the maximum number of cores to use is one less than the total number of cores available.

`quiet`	logical indicating whether feedback should be printed to the console.
`verbose`	logical indicating whether extra feedback should be printed to the console, including progress at each split.
`numcode, frame`	passed to `translate`. Set to NULL (default) unless learning a hybrid DNA/amino acid sequence classifier.
`...`	further arguments to be passed on to `train`).

Details

The "insect" object type is a dendrogram with several additional attributes stored at each node. These include: "clade" the index of the node (see further details below); "sequences" the indices of the sequences in the reference database used to create the object; "taxID" the taxonomic identifier of the lowest common taxon of the sequences belonging to the node (linking to `"db"`); "minscore" the lowest likelihood among the training sequences given the profile HMM stored at the node; "minlength" the minimum length of the sequences belonging to the node; "maxlength" the maximum length of the sequences belonging to the node; "model" the profile HMM derived from the sequence subset belonging to the node; "nunique" the number of unique sequences belonging to the node; "ntotal" the total number of sequences belonging to the node (including duplicates); "key" the hash key used for exact sequence matching (bypasses the classification procedure if an exact match is found; root node only); "taxonomy" the taxonomy database containing the taxon ID numbers (root node only).

The clade indexing system used here is based on character strings, where "0" refers to the root node, "01" is the first child node, "02" is the second child node, "011" is the first child node of the first child node, etc. The leading zero may be omitted for brevity. Note that each inner node can not have more than 9 child nodes.

Value

an object of class `"insect"`.

Author(s)

Shaun Wilkinson

References

Blackshields G, Sievers F, Shi W, Wilm A, Higgins DG (2010) Sequence embedding for fast construction of guide trees for multiple sequence alignment. *Algorithms for Molecular Biology*, **5**, 21.

Durbin R, Eddy SR, Krogh A, Mitchison G (1998) Biological sequence analysis: probabilistic models of proteins and nucleic acids. Cambridge University Press, Cambridge, United Kingdom.

Gerstein M, Sonnhammer ELL, Chothia C (1994) Volume changes in protein evolution. *Journal of Molecular Biology*, **236**, 1067-1078.

Juang B-H, Rabiner LR (1990) The segmental K-means algorithm for estimating parameters of hidden Markov models. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, **38**, 1639-1641.

Examples

```
data(whales)
data(whale_taxonomy)
## use all sequences except first one to train the classifier
set.seed(999)
tree <- learn(whales[-1], db = whale_taxonomy, maxiter = 5, cores = 2)
## find predicted lineage for first sequence
classify(whales[1], tree)
## compare with actual lineage
taxID <- as.integer(gsub(".+\\|", "", names(whales)[1]))
get_lineage(taxID, whale_taxonomy)
```

`manipulate` *Further bit-level manipulation of DNA and amino acid sequences.*

Description

These functions provide additional methods to manipulate objects of class "DNAbin" and "AAbin" to supplement those available in the ape package.

Usage

```
## S3 method for class 'DNAbin'
duplicated(x, incomparables = FALSE, pointers = TRUE, ...)

## S3 method for class 'DNAbin'
unique(x, incomparables = FALSE, attrs = TRUE, drop = FALSE, ...)

## S3 method for class 'DNAbin'
subset(x, subset, attrs = TRUE, drop = FALSE, ...)

## S3 method for class 'AAbin'
duplicated(x, incomparables = FALSE, pointers = TRUE, ...)

## S3 method for class 'AAbin'
unique(x, incomparables = FALSE, attrs = TRUE, drop = FALSE, ...)

## S3 method for class 'AAbin'
subset(x, subset, attrs = TRUE, drop = FALSE, ...)
```

Arguments

`x`	a "DNAbin" or "AAbin" object.
`incomparables`	placeholder, not currently functional.

pointers	logical indicating whether the re-replication index key should be returned as a "pointers" attribute of the output vector (only applicable for duplicated.DNAbin and duplicated.AAbin). Note that this can increase the computational time for larger sequence lists.
...	further arguments to be passed between methods.
attrs	logical indicating whether the attributes of the input object whose length match the object length (or number of rows if it is a matrix) should be retained and subsetted along with the object. This is useful if the input object has species, lineage and/or taxon ID metadata that need to be retained following the duplicate analysis.
drop	logical; indicates whether the input matrix (assuming one is passed) should be reduced to a vector if subset to a single sequence. Defaults to FALSE in keeping with the style of the ape package functions.
subset	logical vector giving the elements or rows to be kept.

Value

unique and subset return a DNAbin or AAbin object. duplicated returns a logical vector.

Author(s)

Shaun Wilkinson

References

Paradis E, Claude J, Strimmer K, (2004) APE: analyses of phylogenetics and evolution in R language. *Bioinformatics* **20**, 289-290.

Paradis E (2007) A bit-level coding scheme for nucleotides. https://emmanuelparadis.github.io/misc/BitLevelCodingScheme_20April2007.pdf.

Paradis E (2012) Analysis of Phylogenetics and Evolution with R (Second Edition). Springer, New York.

Examples

```
data(whales)
duplicates <- duplicated.DNAbin(whales, point = TRUE)
attr(duplicates, "pointers")
## returned indices show that the last sequence is
## identical to the second one.
## subset the reference sequence database to only include unques
whales <- subset.DNAbin(whales, subset = !duplicates)
## this gives the same result as
unique.DNAbin(whales)
```

`prune_taxonomy` *Prune taxonomy database.*

Description

This function prunes the taxon database, removing specified taxa as desired to improve speed and memory efficiency.

Usage

```
prune_taxonomy(db, taxIDs, keep = TRUE)
```

Arguments

`db`	a valid taxonomy database, e.g. obtained by running the `taxonomy` function.
`taxIDs`	the names or taxon ID numbers of the taxa to be retained (or discarded if `keep = FALSE`).
`keep`	logical, indicates whether the specified taxa should be kept and the rest of the database removed or vice versa. Defaults to TRUE.

Details

TBA

Value

a data.frame with the same column names as the input object ("taxID", "parent_taxID", "rank", "name").

Author(s)

Shaun Wilkinson

References

Federhen S (2012) The NCBI Taxonomy database. *Nucleic Acids Research* **40**, D136-D143. doi:10.1093/nar/gkr1178.

https://www.ncbi.nlm.nih.gov/taxonomy/

Examples

```
## remove Odontoceti suborder from cetacean taxonomy database
data(whale_taxonomy)
prune_taxonomy(whale_taxonomy, taxIDs = 9722, keep = FALSE)
```

purge *Identify and remove erroneous reference sequences.*

Description

This function evaluates a DNA reference database (a "DNAbin" object) and removes any sequences whose taxonomic metadata appear to be inconsistent with those of their most closely related sequences.

Usage

```
purge(x, db, level = "order", confidence = 0.8, cores = 1, quiet = FALSE, ...)
```

Arguments

x	a DNAbin list object whose names include taxonomic identification numbers (see searchGB for details).
db	a valid taxonomy database containing the taxonomic identification numbers included in the "names" attribute of the primary input object (a data.frame object; see taxonomy).
level	character string giving the taxonomic level at which heterogeneity within a cluster will flag a sequence as potentially erroneous. This should be a recognized rank within the taxonomy database.
confidence	numeric, the minimum confidence value for a sequence to be purged. For example, if confidence = 0.8 (the default value) a sequence will only be purged if its taxonomy differs from at least four other independent sequences in its cluster.
cores	integer giving the number of processors for multithreading. Defaults to 1. This argument may alternatively be a 'cluster' object, in which case it is the user's responsibility to close the socket connection at the conclusion of the operation, for example by running parallel::stopCluster(cores). The string 'autodetect' is also accepted, in which case the maximum number of cores to use is one less than the total number of cores available. Note that in this case there may be a tradeoff in terms of speed depending on the number and size of sequences to be processed, due to the extra time required to initialize the cluster.
quiet	logical indicating whether progress should be printed to the console.
...	further arguments to pass to the otu function in the kmer package (not including nstart).

Details

This function first clusters the sequence dataset into operational taxonomic units (OTUs) based on a given genetic similarity threshold. Each cluster is then checked for taxonomic homogeneity at a given rank, and any sequences that appear out of place are removed. The criteria for sequence removal are that at least two other independent studies should contradict the taxonomic metadata attributed to the sequence.

Value

a "DNAbin" object.

Author(s)

Shaun Wilkinson

Examples

```
data(whales)
data(whale_taxonomy)
whales <- purge(whales, db = whale_taxonomy, level = "species",
                threshold = 0.97, method = "farthest")
```

---

qfilter — *Quality filtering for amplicon sequences.*

Description

This function performs several quality checks for FASTQ input files, removing any sequences that do not conform to the specified quality standards. This includes an average quality score assessment, size selection, singleton removal (or an alternative minimum count) and ambiguous base-call filtering.

Usage

```
qfilter(
  x,
  minqual = 30,
  maxambigs = 0,
  mincount = 2,
  minlength = 50,
  maxlength = 500
)
```

Arguments

`x`	a vector of concatenated strings representing DNA sequences (in upper case) or a DNAbin list object with quality attributes. This argument will usually be produced by `readFASTQ`.
`minqual`	integer, the minimum average quality score for a sequence to pass the filter. Defaults to 30.
`maxambigs`	integer, the maximum number of ambiguities for a sequence to pass the filter. Defaults to 0.
`mincount`	integer, the minimum acceptable number of occurrences of a sequence for it to pass the filter. Defaults to 2 (removes singletons).
`minlength`	integer, the minimum acceptable sequence length. Defaults to 50.
`maxlength`	integer, the maximum acceptable sequence length. Defaults to 500.

Value

an object of the same type as the primary input argument (i.e. a "DNAbin" object if x is a "DNAbin" object, or a vector of concatenated character strings otherwise).

Author(s)

Shaun Wilkinson

Examples

```
## download and extract example FASTQ file to temporary directory
td <- tempdir()
URL <- "https://www.dropbox.com/s/71ixehy8e51etdd/insect_tutorial1_files.zip?dl=1"
dest <- paste0(td, "/insect_tutorial1_files.zip")
download.file(URL, destfile = dest, mode = "wb")
unzip(dest, exdir = td)
x <- readFASTQ(paste0(td, "/COI_sample2.fastq"))
## trim primers from sequences
mlCOIintF <- "GGWACWGGWTGAACWGTWTAYCCYCC"
jgHCO2198 <- "TAIACYTCIGGRTGICCRAARAAYCA"
x <- trim(x, up = mlCOIintF, down = jgHCO2198)
## filter sequences to remove singletons, low quality & short/long reads
x <- qfilter(x, minlength = 250, maxlength = 350)
```

---

rc — *Reverse complement DNA in character string format.*

Description

This function reverse complements a DNA sequence or vector of DNA sequences that are stored as character strings.

Usage

```
rc(z)
```

Arguments

z	a vector of DNA sequences in upper case character string format.

Details

This function accepts only DNA sequences in concatenated character string format, see `complement` in the `ape` package for "DNAbin" input objects, and `comp` in the `seqinr` package for when the input object is a character vector.

Value

a vector of DNA sequences as upper case character strings.

Author(s)

Shaun Wilkinson

Examples

```
rc("TATTG")
```

---

read — *Read FASTA and FASTQ files.*

Description

Text parsing functions for reading sequences in the FASTA or FASTQ format into R.

Usage

```
readFASTQ(file = file.choose(), bin = TRUE)

readFASTA(
  file = file.choose(),
  bin = TRUE,
  residues = "DNA",
  alignment = FALSE
)
```

Arguments

`file`	the name of the FASTA or FASTQ file from which the sequences are to be read.
`bin`	logical indicating whether the returned object should be in binary/raw byte format (i.e. "DNAbin" or "AAbin" objects for nucleotide and amino acid sequences, respectively). If FALSE a vector of named character strings is returned.
`residues`	character string indicating whether the sequences to be read are composed of nucleotides ("DNA"; default) or amino acids ("AA"). Only required for `readFASTA` and if `bin = TRUE`.
`alignment`	logical indicating whether the sequences represent an alignment to be parsed as a matrix. Only applies to `readFASTA`.

Details

Compatibility:: The FASTQ convention is somewhat ambiguous with several slightly different interpretations appearing in the literature. For now, this function supports the Illumina convention for FASTQ files, where each sequence and its associated metadata occupies four line of the text file as follows : (1) the run and cluster metadata preceded by an @ symbol; (2) the sequence itself in capitals without spaces; (3) a single "+" symbol; and (4) the Phred quality scores from 0 to 93 represented as ASCII symbols. For more information on this convention see the Illumina help page here .

Speed and Memory Requirements:: For optimal memory efficiency and compatibility with other functions, it is recommended to store sequences in raw byte format as either DNAbin or AAbin objects. For FASTQ files when bin = TRUE, a vector of quality scores (also in raw-byte format) is attributed to each sequence. These can be converted back to numeric quality scores with `as.integer`. For FASTQ files when bin = FALSE the function returns a vector with each sequence as a concatenated string with a similarly concatenated quality attribute comprised of the same ASCII metacharacters used in the FASTQ coding scheme.

This function can take a while to process larger FASTQ files, a multithreading option may be available in a future version.

Value

Either a vector of character strings (if bin = FALSE), or a list of raw ("DNAbin" or "AAbin") vectors, with each element having a "quality" attribute.

Author(s)

Shaun Wilkinson

References

Bokulich NA, Subramanian S, Faith JJ, Gevers D, Gordon JI, Knight R, Mills DA, Caporaso JG (2013) Quality-filtering vastly improves diversity estimates from Illumina amplicon sequencing. *Nat Methods*, **1**, 57-59.

Illumina help page: https://help.basespace.illumina.com/articles/descriptive/fastq-files/

See Also

`writeFASTQ` and `writeFASTA` for writing sequences to text in the FASTA or FASTQ format. See also `read.dna` in the `ape` package.

Examples

```
## download and extract example FASTQ file to temporary directory
td <- tempdir()
URL <- "https://www.dropbox.com/s/71ixehy8e51etdd/insect_tutorial1_files.zip?dl=1"
dest <- paste0(td, "/insect_tutorial1_files.zip")
download.file(URL, destfile = dest, mode = "wb")
unzip(dest, exdir = td)
x <- readFASTQ(paste0(td, "/COI_sample2.fastq"))
```

`replicate` *Dereplicate and rereplicate sequence datasets.*

Description

These functions are used to extract only the unique sequences from a set of DNA reads, with the ability to rebuild the original sequence set at a later time.

Usage

```
dereplicate(x, cores = 1)

rereplicate(x)
```

Arguments

`x`	a list of sequences in `DNAbin` or `AAbin` format, or a vector of sequences as concatenated upper-case character strings.
`cores`	integer giving the number of processors for multithreading (defaults to 1). This argument may alternatively be a 'cluster' object, in which case it is the user's responsibility to close the socket connection at the conclusion of the operation, e.g. by running `parallel::stopCluster(cores)`. The string 'autodetect' is also accepted, in which case the maximum number of cores to use is one less than the total number of cores available.

Value

either a DNAbin/AAbin object, or a vector of concatenated upper-case character strings, depending on the input object.

Author(s)

Shaun Wilkinson

Examples

```
data(whales)
tmp <- dereplicate(whales)
whales <- rereplicate(tmp)
```

samoa — *Table of marine COI amplicon sequence variants from American Samoa*

Description

This matrix contains counts of COI amplicon sequence variants (ASV) extracted from autonomous reef monitoring structures (ARMS) in Ofu, American Samoa. Unpublished data courtesy of Molly Timmers (NOAA).

Usage

```
samoa
```

Format

a 2 x 16 integer matrix containing abundance counts of 16 ASVs from two sites. This table contains the first 16 rows of the 'seqtab.nochim' output from the DADA2 pipeline (https://benjjneb.github.io/dada2/tutorial.html). The column names contain the DNA sequences of the ASVs, and row names correspond with site codes.

searchGB — *Query the NCBI GenBank database.*

Description

searchGB queries GenBank using the Entrez search utilities, and downloads the matching sequences and/or their accession numbers. A vector of accession numbers can be passed in lieu of a query, in which case the function downloads the matching sequences from GenBank. Internet connectivity is required.

Usage

```
searchGB(
  query = NULL,
  accession = NULL,
  sequences = TRUE,
  bin = TRUE,
  db = "nucleotide",
  taxIDs = TRUE,
  prompt = TRUE,
  contact = NULL,
  quiet = FALSE
)
```

Arguments

query	an Entrez search query. For help compiling Entrez queries see https://www.ncbi.nlm.nih.gov/books/NBK3837/#EntrezHelp.Entrez_Searching_Options and https://www.ncbi.nlm.nih.gov/books/NBK49540/.
accession	an optional vector of GenBank accession numbers to be input in place of a search query. If both query and accession arguments are provided the function returns an error. Currently, a maximum of 200 accession numbers can be processed at a time.
sequences	logical. Should the sequences be returned or only the GenBank accession numbers? Note that taxon IDs are not returned if `sequences` is set to FALSE.
bin	logical indicating whether the returned sequences should be in raw-byte format ("DNAbin" or "AAbin" object type) or as a vector of named character strings. Defaults to TRUE.
db	the NCBI database from which to download the sequences and/or accession names. Accepted options are "nucleotide" (default) and "protein".
taxIDs	logical indicating whether the NCBI taxon ID numbers should be appended to the names of the output object (delimited by a "|" character). Defaults to TRUE.
prompt	logical indicating whether to check with the user before downloading sequences.
contact	an optional character string with the users email address. This is added to the E-utilities URL and may be used by NCBI to contact the user if the application causes unintended issues.
quiet	logical indicating whether the progress should be printed to the console.

Details

This function uses the Entrez e-utilities API to search and download sequences from GenBank. Occasionally users may encounter an unknown non-reproducible error and appears to be related to database records being updated in GenBank. This can generally be remedied by re-running the function. If problems persist please feel free to raise an issue on the package bug-reports page at <https://github.com/shaunpwilkinson/insect/issues>.

Value

a list of sequences as either a `DNAbin` or `AAbin` object (depending on `"db"`), or a named vector of character strings (if `bin = FALSE`).

Author(s)

Shaun Wilkinson

References

NCBI Resource Coordinators (2012) Database resources of the National Center for Biotechnology Information. *Nucleic Acids Research*, **41** (Database issue): D8–D20.

See Also

read.GenBank (ape) for an alternative means of downloading DNA sequences from GenBank using accession numbers.

Examples

```
## Query the GenBank database for Eukaryote mitochondrial 16S DNA sequences
## between 100 and 300 base pairs in length that were modified between
## the years 1999 and 2000.

  query <- "Eukaryota[ORGN]+AND+16S[TITL]+AND+100:300[SLEN]+AND+1999:2000[MDAT]"
  x <- searchGB(query, prompt = FALSE)
```

---

shave *Shave ends from DNA and amino acid sequences*

Description

This function uses the Viterbi algorithm to semi-globally align a motif to a DNA or AA sequence, and removes all nucleotides to the left and/or right of the motif.

Usage

```
shave(x, motif, direction = "both", cores = 1, ...)
```

Arguments

x	an object of class DNAbin or AAbin.
motif	a DNAbin, AAbin or PHMM object.
direction	character string indicating the direction of the shave. Options are "forward" (shaves everything to the right of the motif), "backward" (shaves everything to the left of the motif) or "both" (retains the motif region only).
cores	integer giving the number of processors for multithreading. Defaults to 1, and reverts to 1 if x is not a list. This argument may alternatively be a 'cluster' object, in which case it is the user's responsibility to close the socket connection at the conclusion of the operation, for example by running parallel::stopCluster(cores). The string 'autodetect' is also accepted, in which case the maximum number of cores to use is one less than the total number of cores available. Note that in this case there may be a tradeoff in terms of speed depending on the number and size of sequences to be processed, due to the extra time required to initialize the cluster.
...	further arguments to be passed to Viterbi (not including 'type').

Details

This functions finds the optimal semiglobal alignment (a.k.a. "glocal" alignment or global alignment with free end gaps) between a sequence `"x"` and a shorter sequence `"motif"`, returning the motif region of x along with the nucleotides to the left or right if `direction` is set to `"reverse"` or `"forward"`, respectively.

Value

an object of class `DNAbin` or `AAbin` (depending on the input object).

Author(s)

Shaun Wilkinson

See Also

`virtualPCR`.

Examples

```
data(whales)
motif = char2dna("AAGTGTAGCATCACTTATTGATCCAAATT")
shave(whales, motif = motif, direction = "both")
```

---

stitch *Paired-end read stitching.*

Description

This function aligns forward and reverse reads generated from a 2x amplicon sequencing platform, and produces a consensus sequence with maximum base-call quality scores attached as "quality" attributes.

Usage

```
stitch(x, y, up = NULL, down = NULL, mindiff = 6, minoverlap = 16)
```

Arguments

x, y	DNAbin objects with quality attributes (see `readFASTQ` to generate these objects from fastq text files), representing the forward and reverse reads to be stitched. These objects can be either vectors or lists. In the latter case, the two objects must be equal length.
up, down	forward and reverse primer sequences (either as concatenated character strings or "DNAbin" objects). Either both or neither should be provided (not one or the other).

`mindiff` the minimum difference in quality between two different base calls for the higher quality call to be added to the consensus alignment. In cases where the quality differences are less than this threshold, the ambiguity code "N" is added to the consensus sequence. Defaults to 6.

`minoverlap` integer giving the minimum number of nucleotides that should be shared between the forward and reverse sequence alignments. Defaults to 16.

Value

a "DNAbin" object or a vector of concatenated character strings, depending on the input.

Author(s)

Shaun Wilkinson

See Also

`readFASTQ`.

Examples

```
## download and extract example FASTQ file to temporary directory
td <- tempdir()
URL <- "https://www.dropbox.com/s/71ixehy8e51etdd/insect_tutorial1_files.zip?dl=1"
dest <- paste0(td, "/insect_tutorial1_files.zip")
download.file(URL, destfile = dest, mode = "wb")
unzip(dest, exdir = td)
x <- readFASTQ(paste0(td, "/COI_sample1_read1.fastq"), bin = FALSE)
y <- readFASTQ(paste0(td, "/COI_sample1_read2.fastq"), bin = FALSE)
z <- stitch(x, y)
z[1]
attr(z, "quality")[1]
```

`taxonomy` *Download taxonomy database.*

Description

This function downloads an up-to-date copy of the taxonomy database.

Usage

```
taxonomy(db = "NCBI", synonyms = FALSE)
```

Arguments

db	character string specifying which taxonomy database to download. Currently only "NCBI" is supported.
synonyms	logical indicating whether synonyms should be included. Note that this increases the size of the returned object by around 10%.

Details

This function downloads the specified taxonomy database as a data.frame object with the following columns: "taxID", "parent_taxID", "rank", "name". As of early 2018 the zip archive to download the NCBI taxonomy database is approximately 40Mb in size, and the output dataframe object is around 200Mb in memory. Once downloaded, the dataframe can be pruned for increased speed and memory efficiency using the function `prune_taxonomy`.

Value

a dataframe with the following elements: "taxID", "parent_taxID", "rank", "name".

Author(s)

Shaun Wilkinson

References

Federhen S (2012) The NCBI Taxonomy database. *Nucleic Acids Research* **40**, D136-D143. doi:10.1093/nar/gkr1178.

https://www.ncbi.nlm.nih.gov/taxonomy/

Examples

```
# db <- taxonomy()
```

---

trim — *Trim primer and/or index sequences.*

Description

This function trims the primer and/or index sequence(s) from a set of DNA sequences.

Usage

```
trim(x, up, down = NULL)
```

Arguments

x	a "DNAbin" object or a vector of concatenated upper-case character strings, containing the sequences to be trimmed.
up, down	"DNAbin" objects or vectors of concatenated upper-case character strings, representing the primer sequences.

Details

Any sequences not containing the primer(s) in either direction are discarded. Hence this function can also be used to de-multiplex sequences and remove indices, though it will generally be faster to do this on the sequencing platform prior to exporting the FASTQ files.

Value

a "DNAbin" object or a vector of concatenated character strings, depending on the input.

Author(s)

Shaun Wilkinson

Examples

```
## download and extract example FASTQ file to temporary directory
td <- tempdir()
URL <- "https://www.dropbox.com/s/71ixehy8e51etdd/insect_tutorial1_files.zip?dl=1"
dest <- paste0(td, "/insect_tutorial1_files.zip")
download.file(URL, destfile = dest, mode = "wb")
unzip(dest, exdir = td)
x <- readFASTQ(paste0(td, "/COI_sample2.fastq"))
## trim primers from sequences
mlCOIintF <- "GGWACWGGWTGAACWGTWTAYCCYCC"
jgHCO2198 <- "TAIACYTCIGGRTGICCRAARAAYCA"
x <- trim(x, up = mlCOIintF, down = jgHCO2198)
```

---

virtualFISH *Virtual* in situ *hybridization.*

Description

This function queries a list of DNA sequences with a virtual probe (either a sequence or a profile hidden Markov model) and returns only the sequences and regions that are of sufficient similarity based on log-odds alignment scoring.

Usage

```
virtualFISH(
  x,
  probe,
  minscore = 100,
  minamplen = 50,
  maxamplen = 500,
  up = NULL,
  down = NULL,
  rcdown = TRUE,
  minfsc = 60,
  minrsc = 60,
  maxNs = 0.02,
  cores = 1,
  quiet = FALSE
)
```

Arguments

`x`	a list of DNA sequences in `DNAbin` format.
`probe`	a DNA sequence ("DNAbin" object) or profile hidden Markov model ("PHMM" object) to be used as the virtual hybridization probe.
`minscore`	numeric; the minimum specificity (log-odds score for the optimal alignment) between the query sequence and the probe for the former to be retained in the output object.
`minamplen`, `maxamplen`	integers giving the minimum and maximum acceptable amplicon lengths.
`up`, `down`	optional objects of class `DNAbin` giving the forward and reverse primer sequences with which to query the sequence list following virtual probe hybridization.
`rcdown`	logical indicating whether the reverse primer should be reverse-complemented prior to aligning with the input sequences. Should be set to TRUE if `down` is the reverse complement of the target sequence (e.g. the sequence of a reverse primer as would be ordered from an oligo supplier).
`minfsc`	numeric, giving the minimum specificity(log-odds score for the optimal alignment) between the forward primer and a sequence for that sequence to be retained.
`minrsc`	numeric, the minimum specificity (log-odds score for the optimal alignment) between the reverse primer (if provided) and a sequence for that sequence to be retained.
`maxNs`	numeric giving the maximum acceptable proportion of the ambiguous residue "N" within the output sequences. Defaults to 0.02.
`cores`	integer giving the number of processors for multithreading. Defaults to 1, and reverts to 1 if `x` is not a list. This argument may alternatively be a 'cluster' object, in which case it is the user's responsibility to close the socket connection at the conclusion of the operation, for example by running `parallel::stopCluster(cores)`. The string 'autodetect' is also accepted, in which case the maximum number of

cores to use is one less than the total number of cores available. Note that in this case there may be a tradeoff in terms of speed depending on the number and size of sequences to be processed, due to the extra time required to initialize the cluster.

`quiet` logical indicating whether progress should be printed to the console.

Details

This function is generally used when filtering/trimming a local sequence database, to mop up any high-scoring sequences with partial/missing primer bind sites that were not included in the output of the `virtualPCR`. For example, this includes sequences that were generated using the same primer set as used in the virtual PCR, and whose primer binding sites were trimmed prior to deposition in the sequence database. Unlike the virtualPCR function, there is no option to retain the primer-bind sites in the returned sequences.

Value

a list of trimmed sequences, returned as an object of class `DNAbin`.

Author(s)

Shaun Wilkinson

See Also

`virtualPCR`

Examples

```
## ensure whale sequences are globally alignable
data(whales)
model <- aphid::derivePHMM(whales)
z <- virtualFISH(whales, probe = model)
```

`virtualPCR` *Virtual PCR.*

Description

`virtualPCR` queries a list of DNA sequences with virtual primers (either sequences or profile hidden Markov models) and returns only the sequences that contain regions of sufficient similarity based on log-odds alignment scoring.

Usage

```
virtualPCR(
  x,
  up,
  down = NULL,
  rcdown = TRUE,
  trimprimers = FALSE,
  minfsc = 50,
  minrsc = 50,
  minamplen = 50,
  maxamplen = 2000,
  maxNs = 0.02,
  partialbind = TRUE,
  cores = 1,
  quiet = FALSE
)
```

Arguments

`x`	a list of DNA sequences in `DNAbin` format.
`up`	an object of class `DNAbin` or `PHMM` giving the forward primer with which to query the sequence list.
`down`	an optional argument the same type as `up` giving the reverse primer with which to query the sequence list. If NULL only the forward primer is used.
`rcdown`	logical indicating whether the reverse primer should be reverse-complemented prior to aligning with the input sequences. Set to TRUE only if `down` is not NULL, is of class `DNAbin`, and is the reverse complement of the target sequence (e.g. the sequence of a reverse primer as would be ordered from an oligo supplier).
`trimprimers`	logical indicating whether the primer-binding sites should be removed from the sequences in the returned list.
`minfsc`	numeric, giving the minimum specificity(log-odds score for the optimal alignment) between the forward primer and a sequence for that sequence to be retained.
`minrsc`	numeric, the minimum specificity (log-odds score for the optimal alignment) between the reverse primer (if provided) and a sequence for that sequence to be retained.
`minamplen`, `maxamplen`	integers giving the minimum and maximum acceptable amplicon lengths. Sequences are discarded if the number of base pairs between the primer-binding sites falls outside of these limits.
`maxNs`	numeric giving the maximum acceptable proportion of the ambiguous residue "N" within the output sequences. Defaults to 0.02.
`partialbind`	logical indicating whether partial primer matching is accepted. Defaults to TRUE.

cores	integer giving the number of processors for multithreading. Defaults to 1, and reverts to 1 if x is not a list. This argument may alternatively be a 'cluster' object, in which case it is the user's responsibility to close the socket connection at the conclusion of the operation, for example by running `parallel::stopCluster(cores)`. The string 'autodetect' is also accepted, in which case the maximum number of cores to use is one less than the total number of cores available. Note that in this case there may be a tradeoff in terms of speed depending on the number and size of sequences to be processed, due to the extra time required to initialize the cluster.
quiet	logical indicating whether progress should be printed to the console.

Value

a list of trimmed sequences, an object of class `DNAbin`.

Author(s)

Shaun Wilkinson

Examples

```
## trim whale sequences using a new set of inner primers
inner_for <- "CGGTTGGGGTGACCTCGGAGTA"
inner_rev <- "GCTGTTATCCCTAGGGTAA"
whales_short <- virtualPCR(whales, up = inner_for, down = inner_rev,
                           trimprimers = TRUE)
```

`whales` *Cetacean 16S rDNA sequences.*

Description

A dataset containing 19 mitochondrial 16S rDNA sequences from 18 cetacean species, downloaded from GenBank on 27 March 2018.

Usage

```
whales
```

Format

A "DNAbin" list object containing 19 cetacean mitochondrial sequences in raw-byte format, averaging 130 nucleotides in length. The sequences are named with the GenBank accession numbers followed by a "|" symbol, followed by their NCBI taxonomy ID numbers. The sequences were downloaded using the `searchGB` function on 17 June 2018 (query term: "cetacea[ORGN]+AND+16S+rRNA[GENE]"), and trimmed using the `virtualPCR` function with the primers 16Smam1 and 16Smam2 (CGGTTGGGGTGACCTCGGA and GCTGTTATCCCTAGGGTAACT, respectively; Taylor 1996).

Source

https://www.ncbi.nlm.nih.gov/genbank/

References

Taylor PG (1996) Reproducibility of ancient DNA sequences from extinct Pleistocene fauna. *Molecular Biology and Evolution*, **13**, 283-285.

`whale_taxonomy` *Cetacean section of NCBI taxonomy database.*

Description

A copy of the NCBI taxonomy reference database, subsetted to include only the cetacean taxa in the `whales` dataset.

Usage

```
whale_taxonomy
```

Format

A data.frame object with 72 rows and four columns, labeled as follows:

taxID the NCBI unique taxon identifier (integer).

parent_taxID the NCBI unique taxon identifier of the immediate parent taxon (integer).

rank The taxonomic rank (i.e. species, genus, etc; character).

name The scientific name of the taxon (character).

The database was accessed from ftp://ftp.ncbi.nlm.nih.gov/pub/taxonomy/taxdump.tar.gz on 17 June 2018 using the `taxonomy` function, and pruned using `prune_taxonomy` with `taxIDs = as.integer(gsub(".+\|", "", names(whales)))`.

Source

https://www.ncbi.nlm.nih.gov/taxonomy/

`write` *Write sequences to text in FASTA or FASTQ format.*

Description

These functions take a list of DNA or amino acid sequences in `DNAbin` or `AAbin` format and outputs a text file to a specified directory.

Usage

```
writeFASTQ(x, file = "", compress = FALSE, append = FALSE)

writeFASTA(x, file = "", compress = FALSE, append = FALSE, wrap = NULL)
```

Arguments

`x`	a list of sequences in `DNAbin` or `AAbin` format, or a vector of sequences as concatenated upper-case character strings. For writeFASTQ, only DNAbin objects are accepted, and each element should have a vector of quality scores of equal length attributed to the sequence. These vectors are comprised of raw bytes ranging from 00 to 5d (0 to 93 when converted to integers). See `readFASTQ` for more details.
`file`	character string giving a valid file path to output the text to. If file = "" (default setting) the text file is written to the console.
`compress`	logical indicating whether the output file should be gzipped.
`append`	logical indicating whether the output should be appended to the file.
`wrap`	integer giving the maximum number of characters on each sequence line. Defaults to NULL (no wrapping).

Value

NULL (invisibly).

Author(s)

Shaun Wilkinson

References

Illumina help page: `https://help.basespace.illumina.com/articles/descriptive/fastq-files/`

See Also

`readFASTQ` for reading FASTQ files into R, and `write.dna` in the ape package for writing DNA to text in FASTA and other formats.

Examples

```
## download and extract example FASTQ file to temporary directory
td <- tempdir()
URL <- "https://www.dropbox.com/s/71ixehy8e51etdd/insect_tutorial1_files.zip?dl=1"
dest <- paste0(td, "/insect_tutorial1_files.zip")
download.file(URL, destfile = dest, mode = "wb")
unzip(dest, exdir = td)
x <- readFASTQ(paste0(td, "/COI_sample2.fastq"))
## trim primers from sequences
mlCOIintF <- "GGWACWGGWTGAACWGTWTAYCCYCC"
jgHCO2198 <- "TAIACYTCIGGRTGICCRAARAAYCA"
x <- trim(x, up = mlCOIintF, down = jgHCO2198)
## quality filter with size selection and singleton removal
x <- qfilter(x, minlength = 250, maxlength = 350)
## output filtered FASTQ file
writeFASTQ(x, file = paste0(td, "/COI_sample2_filtered.fastq"))
writeFASTA(x, file = paste0(td, "/COI_sample2_filtered.fasta"))
```

Index

∗ **datasets**
samoa, 30
whale_taxonomy, 41
whales, 40

aa2char *(conversion)*, 7
allocateCVI, 2
ape, *7*, *21*, *22*, *26*, *28*
aphid, *10*
as.AAbin, *7*
as.character.AAbin, *7*
as.character.DNAbin, *7*
as.DNAbin, *7*

char2aa *(conversion)*, 7
char2dna *(conversion)*, 7
classify, 3
comp, *26*
complement, *26*
conversion, 7

decodePHMM *(encoding)*, 10
demultiplex, 8
dereplicate *(replicate)*, 29
disambiguate, 9
dna2char *(conversion)*, 7
duplicated.AAbin *(manipulate)*, 21
duplicated.DNAbin *(manipulate)*, 21

encodePHMM *(encoding)*, 10
encoding, 10
expand, 11

get_lineage, 13
get_taxID, 14

hash, 15

insect, 16
insect-package *(insect)*, 16

join, 17

learn, *4–6*, *13*, 17

manipulate, 21

prune_taxonomy, 23, *35*, *41*
purge, 24

qfilter, 25

rc, 26
read, 27
read.dna, *28*
read.GenBank, *32*
readFASTA *(read)*, 27
readFASTQ, *33*, *34*, *42*
readFASTQ *(read)*, 27
replicate, 29
rereplicate *(replicate)*, 29

samoa, 30
searchGB, *18*, *24*, 30, *40*
seqinr, *26*
shave, 32
stitch, 33
subset.AAbin *(manipulate)*, 21
subset.DNAbin, *17*
subset.DNAbin *(manipulate)*, 21

taxonomy, *14*, *15*, *18*, *23*, *24*, 34, *41*
train, *12*, *13*, *19*, *20*
translate, *20*
trim, 35

unique.AAbin *(manipulate)*, 21
unique.DNAbin *(manipulate)*, 21

virtualFISH, 36
virtualPCR, *33*, *38*, 38, *40*
Viterbi, *32*

whale_taxonomy, 41
whales, 40, *41*
write, 42
write.dna, *42*
writeFASTA, *28*
writeFASTA *(*write*)*, 42
writeFASTQ, *28*
writeFASTQ *(*write*)*, 42