

Package: dimoose (via r-universe)

October 9, 2026

Type Package

Title Tools to Reshape and Manipulate Dichotomous Keys Used in
Taxonomic Identification

Version 0.1.0

Description Transforms between dichotomous identification keys and a
variety of phenotype formats, including feature matrices, and
generates interactive dichotomous key wizards. Imports keys
from 'FishBase' <<https://www.fishbase.se/keys/allkeys.php>> and
includes PhyloTree Build 17, the human mitochondrial DNA
phylogeny (van Oven 2015, <[doi:10.1016/j.fsigs.2015.09.155](https://doi.org/10.1016/j.fsigs.2015.09.155)>),
as distributed by 'Haplogrep' 3.

License GPL

Copyright See inst/COPYRIGHTS for third-party data and example pages.

URL <https://github.com/leipzig/dimoose>

BugReports <https://github.com/leipzig/dimoose/issues>

Encoding UTF-8

LazyData true

Depends R (>= 3.5.0)

Imports base64enc, data.tree, httr, R6, rpart, rvest (>= 1.0.0),
stats, utils, withr, xml2

Suggests ape, phangorn, gbm, glmertree, igraph, jpeg, jsonlite, knitr,
lme4, partykit, png, randomForest, reticulate (>= 1.41.0),
rmarkdown, taxize (>= 0.9.100), testthat (>= 3.1.7), tibble,
tidytree

Config/testthat/edition 3

Roxygen list(markdown = TRUE)

VignetteBuilder knitr

Config/roxygen2/version 8.1.0

Config/pak/sysreqs libicu-dev libxml2-dev libssl-dev

Repository <https://leipzig.r-universe.dev>

Date/Publication 2026-10-09 15:29:32 UTC

RemoteUrl <https://github.com/leipzig/dimoose>

RemoteRef HEAD

RemoteSha 8c6b1b67e7d08db01e655068a3fc25de0f079ff1

Contents

alignmentScores	3
arachnida	4
classify	4
dimoose	5
discoverSniglets	7
embedImages	8
embedTexts	9
evaluateKey	9
exampleKeys	10
exportWizard	11
featureScores	12
featureTable	13
generateTree	14
haplotypeMatrix	15
imageMeta	16
imageMime	16
imageSet	17
importFishbase	18
inventNames	19
keyFromAlignment	19
keyFromClusters	21
keyFromLeads	22
keyFromMecit	23
keyFromRpart	24
keyFromSniglets	25
lookKey	26
macroHaplogroup	26
mecit	27
modelMeta	29
parseFishbase	29
patchExemplars	30
phyloree17	31
phyloree17_mutations	32
phyloreeKey	33
rawhtml	34
readHaplogrep	35
recurrentPositions	36
renameSniglets	37
scoreImages	38
simulateExpression	38

alignmentScores

3

snigletNames

40

snigletScores

41

splitVariables

41

toAuspiceJSON

42

toIgraph

42

toNewick

43

toTreedata

44

vibrio

45

visionModel

45

vocabularyPrompts

47

Index

48

alignmentScores	<i>Score aligned sequences against a key built from an alignment</i>
-----------------	--

Description

Computes the tests of a `keyFromAlignment()` key for each sequence, for `classify()`. The sequences must be aligned to the same coordinates the key was built with: the same alignment columns or, for a key built with a reference, an alignment that contains that reference sequence.

Usage

```
alignmentScores(key, alignment, reference = NULL)
```

Arguments

- key

A key from `keyFromAlignment()`.
- alignment

As in `keyFromAlignment()`.
- reference

Sequence to number sites by; by default the reference the key was built with, if any.

Value

A sequences x features matrix: 1 if the sequence has the state the first lead of a couplet asks for, 0 if it has another state, and NA if it has a gap or an ambiguity code there.

arachnida

Key to the Orders of Arachnida

Description

The key from Triplehorn & Johnson (2005), *Borror and DeLong's Introduction to the Study of Insects*, 7th ed., chapter 5, as a table of leads (see [keyFromLeads\(\)](#)); [arachnidaKey\(\)](#) turns it into a [dimoose](#) key. Transcribed by hand from the book.

Usage

```
arachnida
```

Format

A data frame with 22 rows (11 couplets) and 6 columns:

Statement Couplet number.

Choice a for the first lead (1.), b for the second (1').

Character Text of the lead.

Next Couplet the lead leads to, or "-".

Taxon Order the lead ends in, for terminal leads.

Page Page of the book where that order is treated.

Source

Triplehorn CA, Johnson NF (2005). *Borror and DeLong's Introduction to the Study of Insects*, 7th ed. Thomson Brooks/Cole. Transcription: data-raw/arachnida/ in the package sources.

classify

Follow a key by machine

Description

Walks each row of scores through key: at every couplet the lead whose test (Feature, Test, Threshold) passes is taken. When no lead or more than one passes (for instance on a missing score), the result is NA.

Usage

```
classify(key, scores)
```

Arguments

key	A dimoose key with Feature, Test and Threshold lead columns, e.g. from keyFromTerms() .
scores	A numeric matrix, one row per object, with a column for every feature id the key tests (see featureScores() and scoreImages()).

Value

A data frame with id (row names of scores), result and path (the leads taken, e.g. "1a 3b").

dimoose	<i>A dichotomous key</i>
---------	--------------------------

Description

An R6 object holding a dichotomous key as a table of the leads on every path from the root to a terminal taxon. Usually created by [importFishbase\(\)](#) or [parseFishbase\(\)](#). See <https://www.fishbase.se/keys/allkeys.php> for FishBase keys.

Active bindings

desc Key title or description (read-only).

df Data frame with one row per lead (or per term, if terms were separated) per terminal taxon, with columns Statement, Choice, Character, Taxon, and pSt/pCh for the parent lead.

meta Key/value data frame of metadata (citation, image URLs, ...).

taxa Data frame of terminal taxa and their resolved names.

leads Data frame with one row per lead: Statement, Choice, Character, Next (next couplet, or "-" if terminal) and Taxon, plus optional Prev, Image, ImageLink, TaxonUrl, Question, Feature, Test and Threshold. Derived from df when the object was created without one.

features Data frame of features a machine can compute, one per row (see [featureTable\(\)](#)), or NULL for keys meant only for people.

Methods**Public methods:**

- [dimoose\\$new\(\)](#)
- [dimoose\\$print\(\)](#)
- [dimoose\\$summary\(\)](#)
- [dimoose\\$validate\(\)](#)
- [dimoose\\$toDataTree\(\)](#)
- [dimoose\\$toNewick\(\)](#)
- [dimoose\\$clone\(\)](#)

[dimoose\\$new\(\)](#): Create a new dimoose object.

Usage:

```
dimoose$new(df, desc = NA, meta = NA, taxa = NA, leads = NULL, features = NULL)
```

Arguments:

`df` Data frame of leads (see the `df` field).

`desc` Key title or description.

`meta` Key/value metadata data frame.

`taxa` Data frame of terminal taxa.

`leads` Optional data frame of leads (see the `leads` field).

`features` Optional data frame of machine-readable features (see [featureTable\(\)](#)).

`dimoose$print()`: Print a short overview of the key.

Usage:

```
dimoose$print(...)
```

Arguments:

`...` Unused.

`dimoose$summary()`: Counts describing the key.

Usage:

```
dimoose$summary()
```

Returns: A one-row data frame: `title`, `couplets`, `leads`, `taxa`, `depth` (longest path in steps) and `problems` (see [validate\(\)](#)).

`dimoose$validate()`: Check the key: leads to missing couplets, terminal leads without a taxon, couplets with fewer than two leads, couplets that cannot be reached from the first one, and (for keys with features) leads whose machine test is missing or inconsistent.

Usage:

```
dimoose$validate()
```

Returns: A character vector of problems, invisibly if there are none.

`dimoose$toDataTree()`: Convert the key to a [data.tree::Node](#): couplets are internal nodes and taxa leaves; each child node carries the lead label and the character text of the lead that leads to it. A couplet reached from several leads is attached under the first.

Usage:

```
dimoose$toDataTree()
```

`dimoose$toNewick()`: Write the key's topology in Newick format, with couplets as internal node labels and taxa as leaves (labels quoted when needed). A couplet reached from several leads is placed under the first.

Usage:

```
dimoose$toNewick()
```

Returns: A single string ending in `;`.

`dimoose$clone()`: The objects of this class are cloneable with this method.

Usage:

```
dimoose$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

discoverSniglets	<i>Discover sniglets by clustering patch embeddings</i>
------------------	---

Description

A sniglet is a visual feature that a model found and that has no name of its own, so dimoose coins one for it ("dulmzil"). This function runs k-means on every patch embedding of every image and keeps the unit length centroids as the definitions of k sniglets. An image "has" a sniglet when one of its patches is close to that centroid (see [snigletScores\(\)](#)). Once you have looked at what a sniglet picks out, give it a real name with [renameSniglets\(\)](#).

Usage

```
discoverSniglets(patches, k = 40, seed = 1, nstart = 10, exemplars = 4)

discoverTerms(...)
```

Arguments

patches	A numeric array images x regions x dimensions (unit length rows), e.g. <code>embedImages(...)\$patches</code> ; <code>dimnames(patches)[[1]]</code> are the image ids. Rows that are all zero are empty regions and are ignored.
k	Number of sniglets.
seed	Seed for k-means and for inventNames() .
nstart	Random starts for stats::kmeans() .
exemplars	Number of example regions to record per sniglet, each from a different image.
...	Passed to discoverSniglets() .

Details

[discoverTerms\(\)](#), [termScores\(\)](#) and [keyFromTerms\(\)](#) are the earlier names of [discoverSniglets\(\)](#), [snigletScores\(\)](#) and [keyFromSniglets\(\)](#), and still work.

Value

An object of class `dimooseSniglets`: `sniglets` (data frame with `id`, `sniglet` (the coined word), `name` (what people see; the coined word until renamed) and `definition`), `centroids` ($k \times d$, rows named by coined word), `exemplars` (data frame `sniglet`, `image`, `patch`, `score`: the best-matching region of each of the best-matching images), `best` (for every image, the region that matches each sniglet best) and `tiles` (the tiling of the regions, see [embedImages\(\)](#)).

Examples

```
set.seed(1)
patches <- array(rnorm(5 * 9 * 8), c(5, 9, 8))
norms <- sqrt(apply(patches^2, c(1, 2), sum))
patches <- sweep(patches, c(1, 2), norms, "/")
dimnames(patches) <- list(letters[1:5], NULL, NULL)
discoverSniglets(patches, k = 3)$sniglets
```

embedImages	<i>Embed images</i>
-------------	---------------------

Description

Returns one embedding per image and, for `discoverSniglets()`, embeddings of local regions of each image.

Usage

```
embedImages(
  model,
  paths,
  region = NULL,
  patches = TRUE,
  tiles = 4,
  minDetail = 8,
  minFill = 0.5
)
```

Arguments

model	A <code>visionModel()</code> .
paths	Image files, or an <code>imageSet()</code> (its path column is embedded and its id column names the rows).
region	Optional crop <code>c(y0, y1, x0, x1)</code> as fractions, applied to every image before embedding.
patches	If TRUE, also return the local embeddings.
tiles	Size of the grid of tiles: 4 cuts 4 x 4 windows that overlap their neighbours by half. Several sizes give several scales (<code>c(2, 4)</code>). 0 uses the model's patch tokens.
minDetail	Tiles with less detail than this (standard deviation of the grey levels, 0 to 255) are treated as empty background and match nothing.
minFill	In a photo with a plain background (one colour around most of its border), tiles in which the subject covers less than this share are treated as empty too, so that features are about the subject and not about its backdrop. It has no effect on photos without a plain background.

Details

By default the regions are **tiles**: overlapping windows cut out of the image and embedded one by one, as if each were a whole image. A tile's embedding describes only what is inside it, so tiles find parts and surfaces that different images have in common (a stem end, a dimpled peel). Tiles that are mostly plain background are left out. `tiles = 0` uses the model's own patch tokens instead, which is faster (one pass per image) and is what `fordera` does, but in the last layer of a CLIP model each token carries much of the whole image, so the features found are less local.

Value

A list: `image` (images x d matrix, rows named by image id: the file name without extension, or the id of an `imageSet()`) and `patches` (images x regions x d array, or NULL), which remembers the tiling in its `tiles` attribute.

embedTexts	<i>Embed texts</i>
------------	--------------------

Description

Embed texts

Usage

```
embedTexts(model, texts)
```

Arguments

<code>model</code>	A <code>visionModel()</code> .
<code>texts</code>	Character vector.

Value

A `texts` x d matrix with the texts as row names.

evaluateKey	<i>Accuracy of a key followed by machine</i>
-------------	--

Description

Accuracy of a key followed by machine

Usage

```
evaluateKey(key, scores, labels, groups = NULL)
```

Arguments

key	A dimoose key with Feature, Test and Threshold lead columns, e.g. from keyFromTerms() .
scores	A numeric matrix, one row per object, with a column for every feature id the key tests (see featureScores() and scoreImages()).
labels	True label of each row of scores.
groups	Optional named vector mapping labels to coarser groups (e.g. model years to generations) for a second accuracy figure.

Value

A list: results (one row per object with actual, predicted, correct, groupCorrect), accuracy, groupAccuracy and confusion.

exampleKeys	<i>Example keys</i>
-------------	---------------------

Description

Ready-made [dimoose](#) keys built from the datasets that come with dimoose. All four are ordinary dimoose objects: they share the same fields (leads, df, meta, taxa, desc) and methods (print(), summary(), validate(), toDataTree(), toNewick()) and all work with [exportWizard\(\)](#).

Usage

```
sharkKey()

arachnidaKey()

vibrioKey()
```

Details

- `sharkKey()`: FishBase key 1, *Key to the families of sharks in the Western Central Pacific* (Compagno 1998), from saved FishBase pages in `inst/extdata/fishbase/`; 23 couplets, 24 families.
- `arachnidaKey()`: the *Key to the Orders of Arachnida* from Triplehorn & Johnson (2005), the [arachnida](#) dataset; 11 couplets, 11 orders (Schizomida is reached from two couplets).
- `vibrioKey()`: a key generated from the [vibrio](#) consensus matrix of Noguerola & Blanch (2008) with [keyFromRpart\(\)](#). Tests scored + or (+) count as positive and - or (-) as negative; variable (v, d) and not determined (ND) results are treated as missing, and `rpart` routes a species past such a test with surrogate splits, so a species that varies for a test appears on only one side of it. Species the tests cannot separate share a result.
- [phyloTreeKey\(\)](#): PhyloTree Build 17.

Value

A [dimoose](#) object.

References

Compagno LJV (1998). General remarks. In Carpenter KE, Niem VH (eds.), *FAO species identification guide for fishery purposes. The living marine resources of the Western Central Pacific*, vol. 2, pp. 1196-1207. FAO, Rome. FishBase key 1: <https://www.fishbase.se/keys/description.php?keycode=1>.

Triplehorn CA, Johnson NF (2005). *Borror and DeLong's Introduction to the Study of Insects*, 7th ed., chapter 5. Thomson Brooks/Cole. ISBN 0-03-096835-6.

Noguerola I, Blanch AR (2008). Identification of *Vibrio* spp. with a set of dichotomous keys. *Journal of Applied Microbiology* 105, 175-185. [doi:10.1111/j.13652672.2008.03730.x](https://doi.org/10.1111/j.13652672.2008.03730.x)

Examples

```
sharks <- sharkKey()
sharks
summary(arachnidaKey())
vibrioKey()$validate()
## Not run:
exportWizard(sharkKey(), "sharks.html")

## End(Not run)
```

exportWizard

Export an interactive dichotomous key

Description

Writes a dimoose key as a single self-contained HTML page. With JavaScript on, the page steps through the key one couplet at a time, keeps a trail of the choices made (each one clickable to go back to it) and supports the browser's Back button. Without JavaScript, the same page is the full key with every lead hyperlinked to its next couplet or taxon. Each taxon gets a result section listing the characters on its path, and a map of the key shows where the user is (see map).

Usage

```
exportWizard(
  key,
  file = NULL,
  displayLinks = TRUE,
  order = c("original", "parsimony"),
  title = NULL,
  map = TRUE,
  staticPaths = NULL
)
```

Arguments

key	A dimoose object, e.g. from importFishbase() .
file	Path of the HTML file to write. If NULL, the HTML is returned as a string instead.
displayLinks	If TRUE, show the couplet numbers each lead leads to and comes from, as in a printed key. If FALSE, hide them. Leads stay clickable either way.
order	Order of couplets. Only "original" (the key's own order) is implemented.
title	Page title. Defaults to the key's description.
map	If TRUE, include a map of the key: the whole key drawn as a tree, marking the current couplet, the path taken so far and the branches (and number of taxa) still possible. It sits beside the key on wide screens and behind a Map button on phones.
staticPaths	If TRUE, write each taxon's path through the key into the page, so it shows without JavaScript. If FALSE, the page builds the path in the browser when a result is shown, which keeps large keys (such as phylotreeKey() on the whole tree) much smaller. The default is TRUE for keys with up to 1,000 taxa.

Details

The key is checked first: leads pointing at missing couplets, terminal leads without a taxon, couplets with fewer than two leads and couplets that cannot be reached from the first one are reported as warnings and marked in the page.

Value

The path to file, invisibly, or the HTML as a character string if file is NULL.

Examples

```
## Not run:
sharks <- importFishbase(1, usePhyloService = "none")
exportWizard(sharks, "sharks.html")

## End(Not run)
```

featureScores

Compute a key's feature scores from embeddings

Description

Model-free: give it the embeddings and it applies each feature's rule. centroid_patch_max features need patches; clip_text_pair features need image and textEmb (rows named by prompt). For external features supply the scores yourself.

Usage

```
featureScores(key, image = NULL, patches = NULL, textEmb = NULL)
```

Arguments

key	A dimoose key with a features table.
image	objects x d matrix of image embeddings (unit rows).
patches	objects x patches x d array of patch embeddings.
textEmb	prompts x d matrix of text embeddings whose row names are the prompt texts, e.g. from embedTexts() .

Value

An objects x features matrix with feature ids as column names.

See Also

[scoreImages\(\)](#) for images-to-scores in one call.

featureTable	<i>A table of machine-computable features</i>
--------------	---

Description

Keys built from images (see [keyFromTerms\(\)](#) and [keyFromClusters\(\)](#)) carry a features table that says how a machine computes the score each lead tests. [classify\(\)](#) walks a key over a matrix of such scores.

Usage

```
featureTable(  
  id,  
  kind,  
  label = id,  
  prompt = NA,  
  negative_prompt = NA,  
  region = NA,  
  embedding = NULL,  
  exemplars = NULL,  
  definition = NA  
)
```

Arguments

id	Feature ids, e.g. "sniglet:dulmzil" or "text:round_headlights".
kind	"centroid_patch_max" (highest cosine similarity between any image patch embedding and embedding), "clip_text_pair" (cosine with prompt minus cosine with negative_prompt) or "external" (computed by the user).
label	Name shown to people; defaults to id.

prompt, negative_prompt	Texts for clip_text_pair features.
region	Crop as "y0,y1,x0,x1" fractions, or NA for the whole image.
embedding	List of numeric vectors (one per feature), or NULL.
exemplars	List of data frames with columns image, patch, score and optionally png (base64) showing what the feature means.
definition	Optional sentence saying what the feature is, shown in the wizard's glossary (see renameSniglets()).

Value

A data frame with the columns above; embedding and exemplars are list columns.

Examples

```
featureTable("sniglet:x", "centroid_patch_max", embedding = list(c(1, 0, 0)))
```

generateTree	<i>Generate a tree from a feature matrix</i>
--------------	--

Description

Fits a classification tree (or tree ensemble) with the taxon column as the response and every other column as a feature.

Usage

```
generateTree(
  data,
  target_col,
  method = c("rpart", "randomForest", "gbm"),
  params = list()
)
```

Arguments

data	A data frame containing the feature matrix.
target_col	Name of the target/response column.
method	"rpart", "randomForest" (requires the 'randomForest' package), or "gbm" (requires the 'gbm' package; "multinomial" distribution unless the response is numeric 0/1, which uses "bernoulli").
params	Optional list of arguments passed to the fitting function, overriding the defaults.

Value

A fitted rpart, randomForest or gbm object.

Examples

```
features <- data.frame(
  family = factor(c("A", "B", "C", "A", "B", "C")),
  fin = factor(c("x", "y", "y", "x", "y", "y")),
  head = factor(c("p", "p", "q", "p", "p", "q"))
)
generateTree(features, "family",
  params = list(control = rpart::rpart.control(minsplit = 2, minbucket = 1))
)
```

haplotypeMatrix	<i>Which mutations each haplogroup carries</i>
-----------------	--

Description

Follows every haplogroup's path from the root of the tree and records, for each requested site, whether the haplogroup's haplotype differs from the root (RSRS) there. Back-mutations are applied as Haplogrep defines them (they undo the latest change at that site), so a site gained and later reverted on a lineage is not carried below the reversion. Insertions are tracked as their own sites ("573.X", "2491.1"), separate from substitutions at the same position; a deletion counts as carried.

Usage

```
haplotypeMatrix(
  haplogroups,
  sites = NULL,
  tree = dimoose::phylotree17,
  mutations = dimoose::phylotree17_mutations
)
```

Arguments

haplogroups	Haplogroup names, one per row of the result; repeats are allowed (e.g. one per sample).
sites	Sites to report: positions as character ("16311") or insertion sites ("573.X"). Defaults to the sites of recurrentPositions() .
tree, mutations	The tree and its mutations; default phylotree17 and phylotree17_mutations . tree needs haplogroup, parent and depth; mutations needs haplogroup, notation, position, type, ancestral and derived.

Value

An integer matrix of 0/1, one row per entry of haplogroups (row names are the haplogroups) and one column per site.

See Also

[recurrentPositions\(\)](#), [macroHaplogroup\(\)](#), [simulateExpression\(\)](#)

Examples

```
haplotypeMatrix(c("H2a2a1", "U5b", "L0a1b"), sites = c("152", "16311", "16278"))
```

imageMeta	<i>Embed images in a key's metadata</i>
-----------	---

Description

Reads image files and returns image_<id> rows for a key's meta, which `exportWizard()` embeds in the page (each image once); leads refer to them by id in their Image column (several separated by ;).

Usage

```
imageMeta(paths, ids = NULL)
```

Arguments

- paths Image files.
- ids Names for the images; default the file names without extension.

Value

A key/value data frame.

imageMime	<i>MIME type of an image file from its first bytes</i>
-----------	--

Description

MIME type of an image file from its first bytes

Usage

```
imageMime(path)
```

Arguments

- path One file.

Value

"image/png", "image/jpeg", "image/gif", "image/webp" or NA.

imageSet	<i>A set of labelled images</i>
----------	---------------------------------

Description

The starting point for automated key generation: one row per photo, with the class (species, model, ...) the photo shows. A key separates the classes, so there must be at least two.

Usage

```
imageSet(paths, label = NULL)
```

Arguments

paths	Image files, or a single folder. A folder is searched (with its subfolders) for image files: png, jpg/jpeg, gif, webp, bmp and tif/tiff.
label	Class of each image: • NULL: for a folder whose images are in subfolders, the name of the subfolder each image is in (pics/cardinal/IMG_01.jpg is a cardinal); otherwise the file name without its extension; • a character vector as long as paths (for a folder, as long as the files found, which are sorted by path); • a function applied to the file names without extensions (e.g. <code>function(x) sub("_.*\$", "", x)</code> so that 1967_alt counts as 1967).

Value

A data frame with id, path and label. id is the file name without its extension; when two files share a name, their folder is put in front (cardinal_IMG_01) so that ids are unique.

Examples

```
imageSet(c("pics/1967.png", "pics/1967_alt.png"), label = function(x) sub("_.*$", "", x))
imageSet(c("pics/cardinal/1.jpg", "pics/robin/1.jpg"), label = c("cardinal", "robin"))
# the example photos that ship with dimoose: one folder per fruit
pics <- imageSet(system.file("extdata", "pics", package = "dimoose"))
table(pics$label)
```

importFishbase	<i>Import a dichotomous key from FishBase</i>
----------------	---

Description

Downloads a key from FishBase and converts it into a **dimoose** object. See <https://www.fishbase.se/keys/allkeys.php> for the list of keys and their key codes.

Usage

```
importFishbase(
  keycode,
  fishbaseUrl = "https://www.fishbase.se/",
  separateTerms = TRUE,
  usePhyloService = c("GNA", "GNR", "none", "TNRS"),
  embedImages = FALSE
)
```

Arguments

keycode	Numeric or character FishBase key code.
fishbaseUrl	Base URL of the FishBase mirror, with a trailing slash.
separateTerms	If TRUE, split each lead's character statement on ";" into one row per term.
usePhyloService	Taxonomic name resolution: "GNA" verifies terminal taxa with the Global Names verifier (<code>taxize::gna_verifier()</code> , requires the 'taxize' package); "none" skips resolution. "GNR" is accepted as an alias for "GNA". "TNRS" is no longer available because <code>taxize::tnrs()</code> is defunct.
embedImages	If TRUE, download the key's morphology images and store them base64-encoded in meta. Image URLs are always stored.

Value

A **dimoose** object.

See Also

[parseFishbase\(\)](#) to build a key from saved HTML pages.

Examples

```
## Not run:
sharks <- importFishbase(1)
sharks$taxa

## End(Not run)
```

inventNames	<i>Invent pronounceable names</i>
-------------	-----------------------------------

Description

Random words made of consonant onsets, vowels and codas. They name sniglets: visual features that have no English name (as fordera does).

Usage

```
inventNames(n, seed = 1)
```

Arguments

n	Number of names.
seed	Random seed, so the same call gives the same names. The global random number generator state is left unchanged.

Value

A character vector of n distinct lower-case words.

Examples

```
inventNames(5)
```

keyFromAlignment	<i>Build a key from a multiple sequence alignment</i>
------------------	---

Description

One command from aligned sequences to a dichotomous key. A phylogeny program builds a tree from the alignment, the tree is rooted and made strictly bifurcating, and every fork becomes a couplet whose two leads are the alignment sites that tell its two clades apart ("152C 16311T" against "152T 16311C"). Each sequence is a result.

Usage

```
keyFromAlignment(
  alignment,
  tool = c("nj", "bionj", "upgma", "parsimony", "ml", "fasttree", "iqtree"),
  tree = NULL,
  outgroup = NULL,
  reference = NULL,
  type = c("auto", "DNA", "AA"),
  model = NULL,
```

```

    maxSites = 3,
    exe = NULL,
    args = NULL,
    desc = "Key generated from a sequence alignment",
    meta = NULL
)

```

Arguments

alignment	The alignment: a FASTA file, an ape DNABin/AABin object (e.g. from ape::read.dna() for Clustal or PHYLIP files), a phangorn phyDat, a character matrix with one named row per sequence and one column per site, or a named character vector of equal-length strings.
tool	The program that builds the tree: • "nj", "bionj": neighbour joining in ape; • "upgma", "parsimony", "ml": UPGMA, the parsimony ratchet and maximum likelihood in phangorn; • "fasttree", "iqtree": the FastTree or IQ-TREE command-line program, which must be installed.
tree	A tree you built yourself, instead of tool: an ape phylo object, a Newick file or a Newick string, with the sequence names as tip labels. It may cover only some of the sequences, and may have polytomies. This is the way to use any other phylogeny program.
outgroup	Name(s) of the sequence(s) to root on. Without it, a tree that is not already rooted is rooted at its midpoint. Sequences can be named in full or by the first word of their FASTA header.
reference	Name of a sequence in the alignment to number sites by (its ungapped positions; columns where it has a gap are numbered "315.1", "315.2"). By default sites are alignment columns.
type	"DNA", "AA", or "auto" to guess from the sequences.
model	Substitution model, passed to the tool: the model of ape::dist.dna() for the distance methods (default "K80"; "JTT" through phangorn::dist.ml() for proteins), the model of phangorn::pml_bb() for "ml" (default "GTR+G(4)", or "LG+G(4)" for proteins), or IQ-TREE's -m (default: IQ-TREE chooses).
maxSites	Most sites listed in a lead.
exe	Path to the FastTree or IQ-TREE program, if it is not on the PATH.
args	Extra command-line arguments for FastTree or IQ-TREE.
desc, meta	Title and extra metadata of the key.

Details

Sites are chosen per fork: a site is *diagnostic* when the states seen on one side are never seen on the other. Up to maxSites diagnostic sites are listed, those with the least missing data first. If a fork has no diagnostic site, the best-separating one is used and marked (most). If no site separates the two sides, the sequences below it are split by their sites alone, whatever the tree says. Sequences

that no site tells apart share a result ("A / B"), as do identical sequences. Gaps and ambiguity codes are treated as missing data.

The first site of each couplet is also a machine-readable test, so `classify()` can place sequences scored with `alignmentScores()`.

Value

A `dimoose` key. Its meta records the tool, the rooting, the tree in Newick format (newick, names quoted where Newick needs it), how many couplets lack a diagnostic site (`weak_couplets`) and how many do not follow the tree (`regrouped_couplets`).

See Also

`alignmentScores()`, `phylotreeKey()`, `toNewick()`

Examples

```
aln <- c(human = "ACGTACGTACGA", chimp = "ACGTACGTATGA", mouse = "ACCTACATGCCA",
        rat = "ACCTACATGCGT", fish = "TCCTTCGTACCA")
if (requireNamespace("ape", quietly = TRUE)) {
  key <- keyFromAlignment(aln, outgroup = "fish")
  key$leads[, c("Statement", "Character", "Next", "Taxon")]
  classify(key, alignmentScores(key, aln))
}

# the example files that ship with dimoose
fasta <- system.file("extdata", "aligned.fasta", package = "dimoose")
mtdna <- system.file("extdata", "mtdna.fasta", package = "dimoose")
if (requireNamespace("ape", quietly = TRUE)) {
  keyFromAlignment(fasta)
  keyFromAlignment(fasta, tree = system.file("extdata", "aligned.nwk", package = "dimoose"))
  keyFromAlignment(mtdna, reference = "H2a2a1", outgroup = "L0a1")
}
## Not run:
keyFromAlignment(mtdna, tool = "iqtree", outgroup = "L0a1") # needs IQ-TREE installed

## End(Not run)
```

keyFromClusters

Build a key by clustering classes and describing each split

Description

Classes are clustered on the mean embedding of their images (Ward's method), giving a binary tree. Each split is then described by the vocabulary pair that best separates its two sides: the pair maximising $|(L.yes - L.no) - (R.yes - R.no)|$ over the side means, not used above that split. The "Has feature" lead goes to the side the feature scores higher on. Machine answers use the same two prompts.

Usage

```
keyFromClusters(  
  image,  
  images,  
  vocabulary,  
  textEmb,  
  template = "{x}",  
  order = NULL,  
  maxPerSide = 10,  
  calibrate = c("none", "midpoint"),  
  desc = "Key generated from image clusters",  
  meta = NULL  
)
```

Arguments

image	images x d matrix of unit-length image embeddings.
images	imageSet() data frame, one row per row of image.
vocabulary	As in vocabularyPrompts() .
textEmb	Text embeddings of vocabularyPrompts(vocabulary, template) , rows named by prompt (as embedTexts() returns them).
template	Must match the one used to make textEmb.
order	Optional function of a character vector of labels returning a number; at each split the child with the smaller value comes first.
maxPerSide	Most images per side used for the side means.
calibrate	"none" (threshold 0) or "midpoint" (halfway between the two sides' mean scores).
desc, meta	Title and metadata.

Value

A [dimoose](#) key with clip_text_pair features.

keyFromLeads	<i>Build a dimoose key from a table of leads</i>
--------------	--

Description

The common constructor behind [sharkKey\(\)](#), [arachnidaKey\(\)](#), [vibrioKey\(\)](#) and [phylotreeKey\(\)](#): takes one row per lead and returns a [dimoose](#) object.

Usage

```
keyFromLeads(leads, desc = NA, meta = NULL, taxa = NULL, features = NULL)
```

Arguments

leads	Data frame with columns Statement (couplet), Choice, Character (the lead's text), Next (next couplet, or "-" if the lead ends in a result) and Taxon (the result, for terminal leads). Optional columns Label, Image, ImageLink and TaxonUrl are used by <code>exportWizard()</code> ; any others are kept.
desc	Title or description of the key.
meta	Key/value data frame of metadata (e.g. citation).
taxa	Data frame of results with submitted_name and matched_name; by default every taxon matches itself.
features	Optional <code>featureTable()</code> for keys a machine can follow (the leads then also carry Feature, Test and Threshold).

Value

A `dimoose` object.

Examples

```
leads <- data.frame(
  Statement = c("1", "1", "2", "2"), Choice = c("a", "b", "a", "b"),
  Character = c("Wings present", "Wings absent", "Two wings", "Four wings"),
  Next = c("2", "-", "-", "-"), Taxon = c("", "Apterygota", "Diptera", "Other")
)
k <- keyFromLeads(leads, "A tiny key")
k
k$toNewick()
```

keyFromMecit

Turn a mixed-effects tree into a key

Description

Each split becomes a couplet and each terminal node a result, labelled with its stratum level and size. Splits on 0/1 variables read as "Carries 16311" / "Does not carry 16311". The leads carry machine-readable tests on the split variables, so `classify()` can place new samples given their `haplotypeMatrix()`, and `exportWizard()` can draw the tree. Splits on factor columns become text-only couplets ("tissue is liver or lung"), which a person can follow but `classify()` cannot; the function warns when the key has any.

Usage

```
keyFromMecit(fit, desc = NULL, digits = 2)
```

Arguments

fit	A <code>mecit()</code> fit.
desc	Title of the key.
digits	Decimal places for stratum levels.

Value

A [dimoose](#) key.

keyFromRpart

Build a dimoose key from a classification tree

Description

Converts an [rpart::rpart\(\)](#) classification tree into a dichotomous key: every split becomes a couplet with two leads, and every leaf a result. Taxa that the tree cannot separate share a leaf and are listed together ("A / B").

Usage

```
keyFromRpart(
  fit,
  desc = "Key generated from a classification tree",
  meta = NULL
)
```

Arguments

fit	An rpart object with method = "class", e.g. from generateTree() .
desc	Title of the key.
meta	Key/value metadata data frame.

Value

A [dimoose](#) object.

See Also

[vibrioKey\(\)](#)

Examples

```
d <- data.frame(
  taxon = factor(c("A", "B", "C")),
  wings = factor(c("+", "-", "-")),
  spots = factor(c("-", "+", "-"))
)
fit <- generateTree(d, "taxon",
  params = list(control = rpart::rpart.control(minsplit = 2, minbucket = 1, cp = 0)))
keyFromRpart(fit, "Three taxa")$leads
```

keyFromSniglets	<i>Build a key from sniglet scores</i>
-----------------	--

Description

Each step asks whether the image has one sniglet. `method = "balanced"` is fordera's algorithm: a sniglet is present when its score is above the quantile of that sniglet's scores over all images; a label has a sniglet if any of its images does; each split uses the unused sniglet that divides the remaining labels most evenly (ties to the earlier one). `method = "rpart"` suits several images per label. It first gives each sniglet one threshold, the cut of its scores that best separates the labels, and then grows a classification tree on has/lacks, so a sniglet means the same thing wherever the key asks about it.

Usage

```
keyFromSniglets(
  scores,
  images,
  sniglets,
  method = c("balanced", "rpart"),
  quantile = 0.6,
  desc = "Key generated from sniglets",
  meta = NULL
)

keyFromTerms(scores, images, terms, ...)
```

Arguments

<code>scores</code>	images x sniglets matrix from snigletScores() .
<code>images</code>	imageSet() data frame with one row per row of scores.
<code>sniglets</code>	The discoverSniglets() object the scores were computed with. Names given with renameSniglets() are used in the key.
<code>method</code>	"balanced" or "rpart".
<code>quantile</code>	Presence threshold quantile for "balanced".
<code>desc, meta</code>	Title and metadata of the key.
<code>terms, ...</code>	keyFromTerms() is the earlier name of keyFromSniglets() ; its third argument was called <code>terms</code> .

Value

A [dimoose](#) key whose leads carry Feature, Test, Threshold and Question, and whose features table holds the centroids. Feature ids are "sniglet:<coined word>" and never change; label is the name shown, which [renameSniglets\(\)](#) changes.

Both leads describe something

A lead that only says a specimen lacks something gives the reader nothing to look for. Where it can, the "Lacks" lead therefore also names what its images have instead: another sniglet that at least 90% of them have and at most 10% of the other side do ("Lacks gaithiark; has blienvaurk"). The machine still tests only the first sniglet. Every lead also records example regions from the images that take it (Examples), which `patchExemplars()` turns into crops: for a "Lacks" lead with no such second sniglet, they are the regions that come closest to the missing one.

looKey	<i>Leave-one-out evaluation of a key builder</i>
--------	--

Description

Leave-one-out evaluation of a key builder

Usage

```
looKey(images, build, score, groups = NULL)
```

Arguments

images	<code>imageSet()</code> data frame of all images.
build	Function of the training row indices returning a key.
score	Function of (key, test row index) returning a one-row scores matrix for that image.
groups	As in <code>evaluateKey()</code> .

Value

As `evaluateKey()`.

macroHaplogroup	<i>Macrohaplogroup of a haplogroup</i>
-----------------	--

Description

Macrohaplogroup of a haplogroup

Usage

```
macroHaplogroup(
  haplogroup,
  level = c("letter", "depth"),
  depth = 2,
  tree = dimoose::phylotree17
)
```

Arguments

haplogroup	Haplogroup names.
level	"letter" (default) uses the conventional label: L0 to L6 for the African lineages and the leading letter otherwise (H2a is H, U5b is U). This is a naming convention, not a clade: for example D sits inside M phylogenetically. "depth" uses the ancestor at a fixed depth from the root, which is a true clade.
depth	For level = "depth", the depth of the ancestor to report; haplogroups shallower than that are returned as they are.
tree	Default phylotree17 .

Value

A character vector the length of haplogroup.

Examples

```
macroHaplogroup(c("H2a2a1", "L3e1", "U5b"))
macroHaplogroup("H2a", level = "depth", depth = 3)
```

mecit

Mixed-effects conditional inference trees

Description

Fits a tree that partitions observations on candidate split variables (for example which mutations each sample carries, from [haplotypeMatrix\(\)](#)) while a random effect absorbs differences between clusters (for example macrohaplogroups, from [macroHaplogroup\(\)](#)). Without the random effect, a tree also splits on variables that merely track the clusters, such as clade-defining markers; with it, those differences are removed before the tree looks for splits.

Usage

```
mecit(
  y,
  partition,
  cluster = NULL,
  method = c("ctree", "mob"),
  slope = NULL,
  alpha = 0.05,
  minbucket = 20L,
  maxdepth = Inf,
  ranefstart = TRUE,
  abstol = 0.001,
  maxit = 50L
)
```

Arguments

y	Numeric response, one value per observation.
partition	Matrix or data frame of candidate split variables, one row per observation, with column names (which may be non-syntactic, like "16311").
cluster	Cluster of each observation, or NULL for no random effect.
method	"ctree" or "mob".
slope	Optional names of partition columns to give random slopes within clusters, as in (1 + slope cluster).
alpha	Significance level for splitting (Bonferroni-adjusted over the candidate variables).
minbucket	Smallest allowed terminal node.
maxdepth	Maximum tree depth.
ranefstart	If TRUE, start from random effects estimated without a tree.
abstol, maxit	Convergence tolerance and the maximum number of iterations. For "ctree" the tolerance is on the largest change in any observation's random-effect contribution; for "mob" it is passed to <code>glmertree::lmertree()</code> , where it applies to the change in log-likelihood.

Details

method = "ctree" (the default) is the RE-EM algorithm of Sela & Simonoff (2012) with conditional inference trees, as proposed by Fu & Simonoff (2015). It alternates between (1) a `partykit::ctree()` on the response minus the current random-effect contribution and (2) a linear mixed model (`lme4::lmer()`) with one fixed effect per terminal node plus the random effects, until the random effects stop changing. method = "mob" uses model-based partitioning through `glmertree::lmertree()` (Fokkema et al. 2018). In both, ranefstart = TRUE estimates the random effects from an intercept-only mixed model before the first tree, which keeps that tree from splitting on cluster-level variables.

With cluster = NULL the result is a plain conditional inference tree, the baseline a mixed-effects tree is compared with.

Value

An object of class `mecit: tree` (a `partykit` tree), `lmer` (the final mixed model, or NULL), `strata` (data frame of node, mean and n; mean is the stratum level net of random effects), `node` (terminal node of each observation), `ranef` (random-effect contribution of each observation), `method`, `iterations` and `converged`.

References

- Sela RJ, Simonoff JS (2012). RE-EM trees: a data mining approach for longitudinal and clustered data. *Machine Learning* 86, 169-207. doi:10.1007/s1099401152583
- Fu W, Simonoff JS (2015). Unbiased regression trees for longitudinal and clustered data. *Computational Statistics & Data Analysis* 88, 53-74. doi:10.1016/j.csda.2015.02.004
- Fokkema M, Smits N, Zeileis A, Hothorn T, Kelderman H (2018). Detecting treatment-subgroup interactions in clustered data with generalized linear mixed-effects model trees. *Behavior Research Methods* 50, 2016-2034. doi:10.3758/s134280170971x

See Also

`splitVariables()`, `keyFromMecit()`, `simulateExpression()`

Examples

```
## Not run:
sim <- simulateExpression(n = 600, genes = 6)
fit <- mecit(sim$expr[, 1], sim$X, sim$samples$macro)
fit
exportWizard(keyFromMecit(fit), "mecit.html")

## End(Not run)
```

modelMeta

Model details as metadata rows

Description

Model details as metadata rows

Usage

```
modelMeta(model)
```

Arguments

model A `visionModel()`.

Value

A key/value data frame with `model_*` keys.

parseFishbase

Build a dimoose object from FishBase key pages

Description

The offline half of `importFishbase()`: parses a FishBase key description page and its questions (couplet) page, both already downloaded.

Usage

```

parseFishbase(
  description,
  questions,
  separateTerms = TRUE,
  usePhyloService = c("GNA", "GNR", "none", "TNRS"),
  embedImages = FALSE,
  pageUrl = "https://www.fishbase.se/keys/questions.php"
)

```

Arguments

description	The description page (keys/description.php): an xml_document, a file path, or a string of HTML.
questions	The questions page (keys/questions.php), in any of the same forms.
separateTerms	If TRUE, split each lead's character statement on ";" into one row per term.
usePhyloService	Taxonomic name resolution: "GNA" verifies terminal taxa with the Global Names verifier (taxize::gna_verifier(), requires the 'taxize' package); "none" skips resolution. "GNR" is accepted as an alias for "GNA". "TNRS" is no longer available because taxize::tnrs() is defunct.
embedImages	If TRUE, download the key's morphology images and store them base64-encoded in meta. Image URLs are always stored.
pageUrl	URL of the questions page, used to resolve relative image links.

Value

A [dimoose](#) object.

Examples

```

## Not run:
parseFishbase("description.html", "questions.html", usePhyloService = "none")

## End(Not run)

```

patchExemplars

Add example crops to a sniglet key

Description

Cuts out the example regions of each sniglet and of each lead, stores them as PNGs in the key's features and meta, and points the leads at them so [exportWizard\(\)](#) shows them. A "Has" lead shows the sniglet in the images that take that lead; a "Lacks" lead shows what its images have instead (see [keyFromSniglets\(\)](#)). The glossary shows each sniglet's examples across all the images.

Usage

```
patchExemplars(model, key, images, pad = NULL, size = 96)
```

Arguments

<code>model</code>	A visionModel() .
<code>key</code>	A key from keyFromSniglets() .
<code>images</code>	The imageSet() the key was built from (for file paths).
<code>pad</code>	Pixels of context to show around the region. By default none for tiles, and half a patch's width for patch tokens.
<code>size</code>	The crop's output size, px.

Value

The key, modified in place and returned invisibly.

phylotree17

PhyloTree Build 17: the human mitochondrial DNA phylogeny

Description

The 5,435 haplogroups of PhyloTree Build 17 (van Oven 2015) as distributed by Haplogrep 3 (release 17.2 of `phylotree-rsrs-17`), one row per haplogroup, rooted at mtMRCA. Use [phylotreeKey\(\)](#) to turn the tree, or part of it, into an identification key, and [phylotree17_mutations](#) for the mutations one per row.

Usage

```
phylotree17
```

Format

A data frame with 5,435 rows and 5 columns:

haplogroup Haplogroup name.

parent Name of the parent haplogroup (NA for mtMRCA).

depth Number of branches from mtMRCA.

n_subclades Number of direct subclades.

mutations Space-separated mutations of the branch leading to this haplogroup, as written by Haplogrep.

Details

Rows are in the order of Haplogrep's tree.xml, which is preorder (each haplogroup directly before its subclades). Names and mutations are Haplogrep's, verbatim: names such as H2a+152 16311 or K1b1+(16093) label branches that PhyloTree leaves unnamed, and two names (M4"67, M4"67+16311!) contain the Unicode character U+2019. Haplogrep's trailing space on one name is trimmed.

Mutations are in RSRS orientation: each is the state on the branch leading to the haplogroup. 16362C is a substitution (position and derived base), 263G! a back-mutation (it reverts the latest change at that position; !! marks a second reversion), 315.1C or 5899.XC an insertion (X: unspecified length) and 249d a deletion. Unlike some other copies (e.g. MToolBox's), PhyloTree's unstable mutations, which PhyloTree prints in parentheses, are included; the parentheses are not recorded.

Source

Haplogrep 3 tree phylotree-rsrs-17, release 17.2, commit 0e79ddf1b78f7a1b81fb5d607bc31a6ddea7b18b, src/tree.xml (<https://github.com/genepi/phylotree-rsrs-17>), MIT license. The conversion and a script that checks these data against GitHub are in data-raw/phylotree17/ in the package sources.

References

van Oven M (2015). PhyloTree Build 17: Growing the human mitochondrial DNA tree. *Forensic Science International: Genetics Supplement Series* 5, e392-e394. doi:10.1016/j.fsigs.2015.09.155

Schönherr S, Weissensteiner H, Kronenberg F, Forer L (2023). Haplogrep 3 - an interactive haplogroup classification and analysis platform. *Nucleic Acids Research* 51, W263-W268. doi:10.1093/nar/gkad284

See Also

[phylotree17_mutations](#), [phylotreeKey\(\)](#)

Examples

```
head(phylotree17)
phylotree17[phylotree17$parent %in% "H2a2a", ]
```

phylotree17_mutations *PhyloTree Build 17 mutations, one per row*

Description

The mutations of every haplogroup in [phylotree17](#), one row per mutation, in the order Haplogrep lists them, with the bases before and after worked out by following the tree from the RSRS.

Usage

```
phylotree17_mutations
```

Format

A data frame with 13,384 rows and 6 columns:

haplogroup Haplogroup whose branch carries the mutation.

notation The mutation as written by Haplogrep.

position Position in the RSRS/rCRS coordinates.

type transition, transversion, back-mutation, insertion or deletion.

ancestral Base before the mutation (for a reverted insertion, the inserted sequence); NA for insertions.

derived Base after the mutation, or the inserted sequence; NA for deletions and reverted insertions.

Details

Seven back-mutations revert a position that nothing earlier on their path changes (six 16233C!, e.g. in D4c1b1, and D5a2 1438A!); for those the written base is taken as the result.

Source

See [phylotree17](#).

Examples

```
table(phylotree17_mutations$type)
phylotree17_mutations[phylotree17_mutations$haplogroup == "J1c", ]
```

phylotreeKey

Build an identification key from PhyloTree Build 17

Description

Turns [phylotree17](#), or the part of it below root, into a [dimoose](#) key. Each haplogroup with subclades becomes a step at which the user picks the subclade whose defining mutations the sample carries. With paragroups = TRUE every step also offers "none of these", which identifies the paragroup, written as in PhyloTree with a trailing * (e.g. H2a*).

Usage

```
phylotreeKey(root = "mtMRCA", paragroups = TRUE, maxDepth = Inf)
```

Arguments

root	Haplogroup to start from; the default, "mtMRCA", is the whole tree.
paragroups	If TRUE, add a "none of these" choice to every step.
maxDepth	Maximum number of steps below root. Haplogroups at that depth become results even if they have subclades.

Details

The whole tree gives a very large key (2,420 steps and 5,435 possible results with paragroups), so for `exportWizard()` a subtree or a `maxDepth` is usually more practical.

Value

A `dimoose` object whose leads have a `Label` column with the haplogroup names.

See Also

`phylotree17`, `exportWizard()`

Examples

```
h2a <- phylotreeKey("H2a")
head(h2a$leads)
## Not run:
exportWizard(phylotreeKey("U5"), "u5.html")
exportWizard(phylotreeKey(maxDepth = 6), "macrohaplogroups.html")

## End(Not run)
```

rawhtml

Raw FishBase key tables

Description

An R6 object holding a FishBase key's description and its couplet table as downloaded, before conversion to a `dimoose` object.

Active bindings

`desc` Key title (read-only).

`df` The key's couplet table: one row per lead, with columns `Statement`, `Choice`, `Character`, `Next`, `Prev` and `Taxon`.

Methods**Public methods:**

- `rawhtml$print()`
- `rawhtml$new()`
- `rawhtml$clone()`

`rawhtml$print()`: Print the description, table and metadata.

Usage:

`rawhtml$print(...)`

Arguments:

... Unused.

`rawhtml$new()`: Download a FishBase key.

Usage:

```
rawhtml$new(
  keycode,
  fishbaseUrl = "https://www.fishbase.se/",
  separateTerms = TRUE
)
```

Arguments:

`keycode` FishBase key code.

`fishbaseUrl` Base URL of the FishBase mirror.

`separateTerms` Ignored; kept for backward compatibility.

`rawhtml$clone()`: The objects of this class are cloneable with this method.

Usage:

```
rawhtml$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

readHaplogrep

Read a Haplogrep / PhyloTree tree XML

Description

Parses the nested `<haplogroup>` XML that Haplogrep distributes (for example `phyloTree-rcrs-17's tree.xml`) into the same data frame shape as [phyloTree17](#), so any Haplogrep tree can be loaded, not just the bundled snapshot. Each haplogroup's `<details><poly>` entries become its mutations.

Usage

```
readHaplogrep(file, key = FALSE)
```

Arguments

`file` Path to the tree XML.

`key` If TRUE, return a [dimoose](#) key built with `keyFromLeads()` instead of the data frame.

Value

A data frame with `haplogroup`, `parent`, `depth`, `n_subclades` and `mutations`, or a [dimoose](#) object if `key = TRUE`.

See Also

[phylotree17](#), [phylotreeKey\(\)](#)

Examples

```
## Not run:
tree <- readHaplogrep("tree.xml")

## End(Not run)
```

recurrentPositions	<i>Recurrent (homoplastic) mutation sites</i>
--------------------	---

Description

Sites that mutate independently on many branches of the tree, such as 152, 16311 and 16189 in PhyloTree. These are the variants whose effects a tree can find across unrelated lineages.

Usage

```
recurrentPositions(
  min_branches = 10,
  types = c("transition", "transversion", "back-mutation"),
  mutations = dimoose::phylotree17_mutations
)
```

Arguments

min_branches	Minimum number of branches (haplogroups) on which the site mutates.
types	Mutation types to count; by default substitutions and back-mutations. Insertions and deletions are left out, except that a reverted insertion counts as a back-mutation at its insertion site (for example "5899.1").
mutations	Default phylotree17_mutations .

Value

A data frame of site and n_branches, most recurrent first.

Examples

```
head(recurrentPositions())
```

renameSniglets	<i>Rename sniglets</i>
----------------	------------------------

Description

Sniglets start with coined names ("dulmzil") because nobody has looked at them yet. After looking at what a sniglet picks out (the example crops in the wizard's glossary), give it a real name. The coined word stays as the sniglet's permanent identifier, so scores, machine tests and saved name tables keep working; only the text people read changes: the leads ("Has round headlights"), the questions and the glossary.

Usage

```
renameSniglets(x, names = NULL, definitions = NULL)
```

Arguments

x	A key from keyFromSniglets() , or a discoverSniglets() result (so that every key built from it afterwards uses the names).
names	The new names: a named character vector (c(dulmzil = "an egg-crate grille")), a names table as described above, or the path of a CSV file holding one.
definitions	Optional named character vector of definitions, named like names. names may be NULL to change definitions only.

Details

The names table has one row per sniglet you want to rename, with columns sniglet (the coined word, or the sniglet's current name), name and optionally definition. Rules:

- a blank name puts the coined word back;
- no two sniglets may share a name, and a name may not be another sniglet's coined word;
- a definition column replaces the definitions of the rows listed (blank removes one); without the column, definitions are left alone;
- a used column, as [snigletNames\(\)](#) returns, is ignored.

Names read best as noun phrases ("an egg-crate grille"), since leads are written "Has *name*" and "Lacks *name*".

Value

A renamed copy of x; x itself is not changed.

See Also

[snigletNames\(\)](#)

Examples

```
set.seed(1)
patches <- array(rnorm(5 * 9 * 8), c(5, 9, 8))
patches <- sweep(patches, c(1, 2), sqrt(apply(patches^2, c(1, 2), sum)), "/")
s <- discoverSniglets(patches, k = 3)
first <- s$sniglets$sniglet[1]
s <- renameSniglets(s, stats::setNames("round headlights", first),
                    definitions = stats::setNames("Circular lamps in the grille", first))
snigletNames(s)
```

scoreImages	Score new images against a key's features
-------------	---

Description

Embeds the images (once per distinct crop region the key's text features use) and applies `featureScores()`.

Usage

```
scoreImages(key, model, paths)
```

Arguments

key	A key with a features table.
model	A <code>visionModel()</code> .
paths	Image files, or an <code>imageSet()</code> (its path column is embedded and its id column names the rows).

Value

An images x features matrix for `classify()`.

simulateExpression	Simulate expression data on PhyloTree
--------------------	---------------------------------------

Description

Simulates per-sample expression for a set of genes, where recurrent mutations (variants that arise independently on many branches) shift expression by fixed amounts on top of a baseline that differs between macrohaplogroups. It is the ground truth for testing `mecit()`: a good method finds the mutation effects as splits and does not split on clade markers that only track the macrohaplogroup baselines.

Usage

```
simulateExpression(
  n = 1000,
  genes = 20,
  effectSites = c("16362", "150", "709", "16093"),
  propMutation = 0.4,
  propBaseline = 0.3,
  sitesPerGene = 1:2,
  effect = 1,
  macroSD = 1,
  slopeSD = 0,
  noise = 1,
  decoys = NULL,
  macroLevel = "letter",
  seed = 1
)
```

Arguments

n	Number of samples.
genes	Number of genes.
effectSites	Sites that can carry mutation effects.
propMutation, propBaseline	Proportions of mutation and baseline genes; the rest are null.
sitesPerGene	Possible numbers of effect sites per mutation gene.
effect	Size of each mutation effect (its sign is random).
macroSD	Standard deviation of the macrohaplogroup baselines.
slopeSD	If positive, each mutation effect also varies between macrohaplogroups with this standard deviation (a random slope).
noise	Residual standard deviation.
decoys	Extra candidate split sites with no effect. By default the most recurrent other sites plus the markers of the four most common macrohaplogroups in the sample.
macroLevel	Passed to macroHaplogroup() as level.
seed	Random seed.

Details

Samples are drawn uniformly from the leaf haplogroups of [phylotree17](#), so their macrohaplogroups are as unbalanced as the tree itself (H is far more common than L5). For sample i and gene g :

$$\text{expr}[i, g] = b[g, \text{macro}(i)] + \sum_s \text{beta}[g, s] * X[i, s] + \text{slope} + \text{noise}$$

Genes come in three kinds: "mutation" genes have a baseline and one or more mutation effects, "baseline" genes have only the macrohaplogroup baseline, and "null" genes are pure noise.

The default effect sites (16362, 150, 709, 16093) are highly recurrent but carried by a minority of lineages across many macrohaplogroups. The most-cited recurrent sites (152, 16311, 146, 16189)

are poor effect sites in RSRS orientation: they are also on deep branches (16311 is an L3 marker), so most lineages carry them and carrying them largely tracks clade membership. They are included as decoys instead.

Value

A list with `expr` (samples x genes), `samples` (data frame of sample, haplogroup, macro), `X` (samples x sites, 0/1, from `haplotypeMatrix()`), `truth` (beta, genes x sites; b, genes x macrohaplogroups; `geneType`; and `slope` when `slopeSD > 0`), `effectSites`, `markerSites` and `decoySites`.

See Also

`mecit()`, `haplotypeMatrix()`

Examples

```
sim <- simulateExpression(n = 300, genes = 10)
table(sim$truth$geneType)
```

snigletNames	<i>Names of a key's sniglets</i>
--------------	----------------------------------

Description

Lists the sniglets of a key (or of a `discoverSniglets()` result) with the names they currently go by. This is the table `renameSniglets()` takes: write it to a CSV file, fill in the name and definition columns, and read it back.

Usage

```
snigletNames(x)
```

Arguments

`x` A key from `keyFromSniglets()`, or a `discoverSniglets()` result.

Value

A data frame with one row per sniglet:

- `sniglet`: the coined word. It identifies the sniglet and never changes.
- `name`: the name people see. It is the coined word until you rename it.
- `definition`: an optional sentence saying what the sniglet is.
- `used` (keys only): whether the key asks about this sniglet. Only these need names.

See Also

`renameSniglets()`

snigletScores	<i>Score images against sniglets</i>
---------------	--------------------------------------

Description

Score images against sniglets

Usage

```
snigletScores(patches, centroids)
```

```
termScores(patches, centroids)
```

Arguments

patches As in [discoverSniglets\(\)](#).

centroids A $k \times d$ matrix of sniglet centroids (unit rows), e.g. `discoverSniglets(...)$centroids`.

Value

An $\text{images} \times \text{sniglets}$ matrix: for each sniglet, the highest cosine similarity between the centroid and any patch of the image. Columns are named by the coined words, whatever the sniglets have been renamed to.

splitVariables	<i>Variables a fitted tree splits on</i>
----------------	--

Description

Variables a fitted tree splits on

Usage

```
splitVariables(fit)
```

Arguments

fit A [mecit\(\)](#) fit.

Value

The names (as given in partition) of the variables used in any split, in the order they first appear; character(0) if the tree has no splits.

toAuspiceJSON

Export a key as Auspice (Nextstrain) JSON v2

Description

Writes the key's spanning tree as an Auspice v2 dataset, which the Nextstrain viewer and auspice.us can display. Node names are made unique and space-free (required by the schema); the original labels and each lead's text are kept in `node_attrs`. `div` is the depth from the root.

Usage

```
toAuspiceJSON(key, title = NULL, file = NULL)
```

Arguments

<code>key</code>	A dimoose key.
<code>title</code>	Dataset title; defaults to the key's description.
<code>file</code>	Optional path to write to; if NULL, the JSON is returned as a string.

Value

The JSON string (invisibly if written to `file`).

See Also

[toNewick\(\)](#), [toTreedata\(\)](#)

Examples

```
js <- toAuspiceJSON(phyлотreeKey("H2a"))
```

toIgraph

Convert a key to an igraph directed graph

Description

Unlike [toNewick\(\)](#) and [toDataTree\(\)](#), which use a spanning tree, this keeps the key's true structure: a couplet reached from several leads has several incoming edges. Each edge carries the lead's label and Character text; vertices are couplets and taxa.

Usage

```
toIgraph(key)
```

Arguments

key A [dimoose](#) key.

Value

An [igraph::igraph](#) directed graph.

Examples

```
## Not run:
g <- toIgraph(arachnidaKey())
igraph::vcount(g)

## End(Not run)
```

toNewick	<i>Write a key as Newick, with safe labels or node annotations</i>
----------	--

Description

The [dimoose](#) method `$toNewick()` writes quoted Newick, which is correct but which several parsers (ape, scikit-bio, toytree) mangle. This wrapper adds two options for interoperating with the wider phylogenetics ecosystem.

Usage

```
toNewick(
  key,
  labels = c("quoted", "safe"),
  annotate = c("none", "nhx", "beast")
)
```

Arguments

key A [dimoose](#) key.

labels "quoted" (the default, identical to `key$toNewick()`) or "safe", which replaces every label with an ASCII id (n<couplet> for couplets, t<n> for taxa) and returns the mapping in a side table, so fragile parsers never see spaces or punctuation. The real text travels in the companion table (this is also the Newick-plus-table shape Taxonium reads).

annotate "none" (default), "nhx" or "beast": attach each lead's Character text to the node it leads to, as an NHX ([&&NHX:lead=...]) or BEAST-style ([&lead=...]) comment. Values are sanitized (, : ; = " [] and newlines removed) because Newick comments cannot carry them. Only one of labels = "safe" or annotate applies at a time; annotate wins if both are given.

[]: R:%20

Value

For labels = "quoted" or any annotate, a single Newick string. For labels = "safe" with annotate = "none", a list of newick (the string) and labels (a data frame of id and label).

See Also

[toTreedata\(\)](#), [toAuspiceJSON\(\)](#)

Examples

```
k <- arachnidaKey()
cat(toNewick(k, annotate = "nhx"))
```

toTreedata

Convert a key to a tidytree treedata (for ggtree)

Description

Builds a treedata object whose tree is the key's spanning tree and whose node data carries each lead's Character, Choice and Taxon. The data of a child node describes the edge into it, so in ggtree you draw lead text on branches with `geom_label(aes(x = branch, label = Character))` and join your own data by the node column (never by label, since taxa repeat).

Usage

```
toTreedata(key)
```

Arguments

key A [dimoose](#) key.

Value

A [tidytree::treedata](#) object.

See Also

[toNewick\(\)](#), [toIgraph\(\)](#)

Examples

```
## Not run:
td <- toTreedata(arachnidaKey())
library(ggtree)
ggtree(td) + geom_tiplab() + geom_label(aes(x = branch, label = Character))

## End(Not run)
```

vibrio*Vibrio consensus identification matrix*

Description

Biochemical and growth test results for 74 *Vibrio* taxa from Noguerola & Blanch (2008), one row per taxon. `vibrioKey()` generates a key from it.

Usage

```
vibrio
```

Format

A data frame with 74 rows and 46 columns: species, then 45 tests coded as in the paper: + positive, - negative, (+) / (-) mostly positive / negative, v or d variable, ND not determined (and one blank cell in the source, NA). Tests under the paper's "Acid from" and "Resistance to" headings are named with that prefix.

Source

Noguerola I, Blanch AR (2008). Identification of *Vibrio* spp. with a set of dichotomous keys. *Journal of Applied Microbiology* 105, 175-185. doi:10.1111/j.13652672.2008.03730.x. Extracted from the paper's table; see data-raw/vibrio/ in the package sources.

visionModel*Load a vision model for embedding images*

Description

Uses Python's open_clip through reticulate. You do not need to install anything in Python yourself: the first call sets up a private Python with the packages dimoose needs (torch, open_clip_torch, pillow, numpy) and reuses it afterwards. That first call downloads them, which takes a few minutes. This needs reticulate 1.41 or later.

Usage

```
visionModel(  
  weights = NULL,  
  name = "ViT-B-32",  
  pretrained = "openai",  
  python = NULL  
)
```

Arguments

weights	Local checkpoint file: OpenAI's ViT-B-32.pt, or an open_clip checkpoint for the architecture name. If NULL, the weights are downloaded.
name	The model: an open_clip architecture name, "bioclip", "bioclip-2", "hf-hub:<organisation>/<model>" or "local-dir:<folder>" (see below).
pretrained	open_clip's tag for the weights to download for an architecture name; not used for the other kinds of name.
python	Optional path to the Python executable to use (passed to <code>reticulate::use_python()</code>).

Details

To use a Python of your own instead, give its path in `python`, or select it with `reticulate::use_python()`, `reticulate::use_virtualenv()` or the RETICULATE_PYTHON environment variable before the first call, and install the packages there with `reticulate::py_install(c("torch", "open_clip_torch", "pillow", "numpy"))`.

Value

An object of class `dimooseVisionModel` with `py` (the Python model) and `spec` (library, architecture, weights SHA-256, image size, ...).

Other models, including BioCLIP

Any open_clip model whose image side is a vision transformer works. The key builders only see its embeddings, so nothing else changes.

- `visionModel(name = "bioclip")` loads **BioCLIP** and `visionModel(name = "bioclip-2")` BioCLIP 2, models trained on images of organisms, from the Hugging Face Hub. They are downloaded on first use.
- `name = "hf-hub:<organisation>/<model>"` loads any other open_clip model on the Hub, and `name = "local-dir:<folder>"` a copy you have downloaded (the folder holds the model's `open_clip_config.json` and weights file).
- `name` and `pretrained` together select one of open_clip's built-in models, e.g. `name = "ViT-B-16"`, `pretrained = "la`

A key can only be followed with the model it was built with: use the same model for `embedImages()` and later for `scoreImages()`.

BioCLIP's text side was trained on names of organisms, not on descriptions of what they look like. It suits sniglets (`discoverSniglets()`), which use images only, better than vocabulary questions (`keyFromClusters()`).

Examples

```
## Not run:
model <- visionModel()           # CLIP ViT-B/32
model <- visionModel(name = "bioclip") # BioCLIP, for photos of organisms

## End(Not run)
```

vocabularyPrompts	<i>Prompts for a vocabulary of visual features</i>
-------------------	--

Description

Prompts for a vocabulary of visual features

Usage

```
vocabularyPrompts(vocabulary, template = "{x}")
```

Arguments

vocabulary	Data frame with feature and opposite columns (and an optional region, "y0,y1,x0,x1" fractions).
template	Text with {x} where the feature goes, e.g. "a pickup truck with {x}".

Value

A character vector: the feature prompt then the opposite prompt for each row, in order. Embed these with `embedTexts()` and pass the result to `keyFromClusters()`.

Examples

```
v <- data.frame(feature = c("round lights", "a flat hood"),
                opposite = c("square lights", "a curved hood"))
vocabularyPrompts(v, "a truck with {x}")
```

Index

* datasets

- arachnida, [4](#)
- phylotree17, [31](#)
- phylotree17_mutations, [32](#)
- vibrio, [45](#)

alignmentScores, [3](#)
alignmentScores(), [21](#)
ape::dist.dna(), [20](#)
ape::read.dna(), [20](#)
arachnida, [4](#), [10](#)
arachnidaKey (exampleKeys), [10](#)
arachnidaKey(), [4](#), [22](#)

classify, [4](#)
classify(), [3](#), [13](#), [21](#), [23](#), [38](#)

data.tree::Node, [6](#)
dimoose, [4](#), [5](#), [10–13](#), [18](#), [21–25](#), [30](#), [33–35](#),
[42–44](#)
discoverSniglets, [7](#)
discoverSniglets(), [7](#), [8](#), [25](#), [37](#), [40](#), [41](#), [46](#)
discoverTerms (discoverSniglets), [7](#)

embedImages, [8](#)
embedImages(), [7](#), [46](#)
embedTexts, [9](#)
embedTexts(), [13](#), [22](#), [47](#)
evaluateKey, [9](#)
evaluateKey(), [26](#)
exampleKeys, [10](#)
exportWizard, [11](#)
exportWizard(), [10](#), [16](#), [23](#), [30](#), [34](#)

featureScores, [12](#)
featureScores(), [5](#), [10](#), [38](#)
featureTable, [13](#)
featureTable(), [5](#), [6](#), [23](#)

generateTree, [14](#)
generateTree(), [24](#)

glmertree::lmertree(), [28](#)

haplotypeMatrix, [15](#)
haplotypeMatrix(), [23](#), [27](#), [40](#)

igraph::igraph, [43](#)
imageMeta, [16](#)
imageMime, [16](#)
imageSet, [17](#)
imageSet(), [8](#), [9](#), [22](#), [25](#), [26](#), [31](#), [38](#)
importFishbase, [18](#)
importFishbase(), [5](#), [12](#), [29](#)
inventNames, [19](#)
inventNames(), [7](#)

keyFromAlignment, [19](#)
keyFromAlignment(), [3](#)
keyFromClusters, [21](#)
keyFromClusters(), [13](#), [46](#), [47](#)
keyFromLeads, [22](#)
keyFromLeads(), [4](#), [35](#)
keyFromMecit, [23](#)
keyFromMecit(), [29](#)
keyFromRpart, [24](#)
keyFromRpart(), [10](#)
keyFromSniglets, [25](#)
keyFromSniglets(), [7](#), [30](#), [31](#), [37](#), [40](#)
keyFromTerms (keyFromSniglets), [25](#)
keyFromTerms(), [5](#), [10](#), [13](#)

lme4::lmer(), [28](#)
lookKey, [26](#)

macroHaplogroup, [26](#)
macroHaplogroup(), [15](#), [27](#), [39](#)
mecit, [27](#)
mecit(), [23](#), [38](#), [40](#), [41](#)
modelMeta, [29](#)

parseFishbase, [29](#)
parseFishbase(), [5](#), [18](#)

partykit::ctree(), 28
patchExemplars, 30
patchExemplars(), 26
phangorn::dist.ml(), 20
phangorn::pml_bb(), 20
phylotree17, 15, 27, 31, 32–36, 39
phylotree17_mutations, 15, 31, 32, 32, 36
phylotreeKey, 33
phylotreeKey(), 10, 12, 21, 22, 31, 32, 36

rawhtml, 34
readHaplogrep, 35
recurrentPositions, 36
recurrentPositions(), 15
renameSniglets, 37
renameSniglets(), 7, 14, 25, 40
reticulate::use_python(), 46
reticulate::use_virtualenv(), 46
rpart::rpart(), 24

scoreImages, 38
scoreImages(), 5, 10, 13, 46
sharkKey (exampleKeys), 10
sharkKey(), 22
simulateExpression, 38
simulateExpression(), 15, 29
snigletNames, 40
snigletNames(), 37
snigletScores, 41
snigletScores(), 7, 25
splitVariables, 41
splitVariables(), 29
stats::kmeans(), 7

termScores (snigletScores), 41
tidytree::treedata, 44
toAuspiceJSON, 42
toAuspiceJSON(), 44
toIgraph, 42
toIgraph(), 44
toNewick, 43
toNewick(), 21, 42, 44
toTreedata, 44
toTreedata(), 42, 44

vibrio, 10, 45
vibrioKey (exampleKeys), 10
vibrioKey(), 22, 24, 45
visionModel, 45
visionModel(), 8, 9, 29, 31, 38
vocabularyPrompts, 47
vocabularyPrompts(), 22