Package 'revtools'

December 17, 2019

Version 0.4.1

Date 2019-12-17

Title Tools to Support Evidence Synthesis

Description Researchers commonly need to summarize scientific information, a process known as 'evidence synthesis'. The first stage of a synthesis process (such as a systematic review or meta-analysis) is to download a list of references from academic search engines such as 'Web of Knowledge' or 'Scopus'. The traditional approach to systematic review is then to sort these data manually, first by locating and removing duplicated entries, and then screening to remove irrelevant content by viewing titles and abstracts (in that order). 'revtools' provides interfaces for each of these tasks. An alternative approach, however, is to draw on tools from machine learning to visualise patterns in the corpus. In this case, you can use 'revtools' to render ordinations of text drawn from article titles, keywords and abstracts, and interactively select or exclude individual references, words or topics.

Depends R (>= 3.5.0)

Imports ade4, modeltools, ngram, plotly, slam, shiny, shinydashboard, SnowballC, stringdist, tm, topicmodels, viridisLite

License GPL-3

URL https://revtools.net

BugReports https://github.com/mjwestgate/revtools/issues

LazyData true

NeedsCompilation no

Author Martin J. Westgate [aut, cre]

Maintainer Martin J. Westgate <martinjwestgate@gmail.com>

Repository CRAN

Date/Publication 2019-12-17 07:40:02 UTC

R topics documented:

---

revtools-package *revtools: Tools to support reviews and evidence synthesis*

Description

Researchers commonly need to summarize scientific information, a process known as 'evidence synthesis'. The first stage of a synthesis process (such as a systematic review or meta-analysis) is to download a list of references from academic search engines such as 'Web of Knowledge' or 'Scopus'. The traditional approach to systematic review is then to sort these data manually, first by locating and removing duplicated entries, and then screening to remove irrelevant content by viewing titles and abstracts (in that order). 'revtools' provides interfaces for each of these tasks. An alternative approach, however, is to draw on tools from machine learning to visualise patterns in the corpus. In this case, you can use 'revtools' to render ordinations of text drawn from article titles, keywords and abstracts, and interactively select or exclude individual references, words or topics.

Functions

Article screening

- `screen_duplicates` Screen for duplicates
- `screen_titles` Screen articles by title
- `screen_abstracts` Screen articles by abstract

- screen_topics Screen data by topic

Bibliographic methods

- read_bibliography Import bibliographic data
- write_bibliography Export bibliographic data
- bibliography-class Format for storing bibliographic data
- bibliography-methods Print, summary, as.bibliography, as.data.frame and [methods for class 'bibliography'
- tag_lookup Lookup table for ris tag transformations
- merge_columns rbind two data.frames with different numbers of columns
- format_citation Return a clean citation from a bibliography or data.frame
- add_line_breaks Set a maximum character width for strings in a vector

Distributing tasks among a team

- allocate_effort Specify how to distribute screening effort among a team of reviewers
- distribute_tasks Split a dataset among a team of reviewers
- aggregate_tasks Combine screening results from a team of reviewers

Duplicate detection

- fuzz_functions Fuzzy string matching
- find_duplicates Locate potentially duplicated references
- extract_unique_references Return a data.frame with only 'unique' references
- screen_duplicates Screen for duplicates

Text mining

- revwords Stopwords used in revtools functions
- make_dtm Construct a Document-Term Matrix from bibliographic data
- run_topic_model Wrapper function for topic models
- screen_topics Screen data by topic

---

add_line_breaks *Add line breaks to one or more strings*

Description

This function takes a vector of strings and adds line breaks every n characters. Primarily built to be called internally by format_citation, this function has been made available as it can be useful in other contexts.

Usage

```
add_line_breaks(x, n = 50, html = FALSE)
```

Arguments

`x`	Either a string or a vector; if the vector is not of class `character` if will be coerced to one using `as.character`.
`n`	Numeric: The number of characters that should separate consecutive line breaks.
`html`	Logical: Should the function add HTML line breaks (`<br>`)? Defaults to FALSE, in which case the newline sequence is used instead (escape-n).

Details

Line breaks are only added between words, so the value of `n` is acutally a threshold value rather than being matched exactly.

Value

Returns the input vector unaltered except for the addition of line breaks.

Examples

```
file_location <- system.file(
  "extdata",
  "avian_ecology_bibliography.ris",
  package = "revtools")
x <- read_bibliography(file_location)
add_line_breaks(x$title)
```

`aggregate_tasks` *Combine (potentially overlapping) article sets generated by screening among a team of reviewers.*

Description

A common task in systematic review is to divide a dataset of articles located by a search (typically involving >1 databases) and distributing them amongst a team of reviewers for screening. This function takes a dataset divided using `link{distribute_tasks}` and recombines them into a single `data.frame`.

Usage

```
aggregate_tasks(file_names, match_column, selection_column, reviewer_names)
```

Arguments

`file_names`	a vector or list of file paths used to locate screened files. Must be in .csv format.
`match_column`	The name of the column used to match identical references. In revtools this is 'label', which is the default here.

`selection_column`
: The name of the column used to store 'selection' data; i.e. which entries have been retained and which excluded. In revtools this is 'selected', which is the default here.

`reviewer_names`
: Optional vector of names used to label the 'results' columns in the resulting `data.frame`.

Value

Returns a data.frame with one row per unique value of `match_column`, showing the content of `selection_column` for each reviewer.

See Also

`distribute_tasks` for the inverse problem of dividing a single dataset amongst many reviewers.

allocate_effort *Determine optimal way to divide articles among 2 or more reviewers*

Description

This function takes information on the number (and optionally, identity) of reviewers and combines it with data on how it should be allocated to return a data.frame showing the proportion of articles to be assessed by each person.

Usage

```
allocate_effort(reviewers, effort, proportion_checked,
    max_reviewers = 3, precision = 2, quiet = TRUE)
```

Arguments

`reviewers`
: Either an integer giving the number of reviewers, or a vector of strings giving reviewer names.

`effort`
: Either a single number giving the proportion of articles to be reviewed by all reviewers, or a numeric vector giving a unique proportion for each reviewer. If the latter must be consistent with number given by 'reviewers' above.

`proportion_checked`
: Numeric value giving the proportion of entries that should be screened by two or more reviewers.

`max_reviewers`
: the maximum number of reviewers that should screen any single article. Useful for avoiding redundancy when working with large teams.

`precision`
: Number of decimal places with which to report results. Defaults to 2.

`quiet`
: Logical - should the function return a summary of the proportion of articles allocated to each reviewer? Defaults to TRUE.

Details

This function makes an attempt to return a sensible distribution of effort among a number of reviewers. If effort is given as a single value (or not provided at all), then the result is calculated exactly such that all proportions sum to 1. Conversely, if effort is given as a numeric vector of length >1 and contains a range of values, then the function tries to optimize the proportion of articles allocated to each person, while also matching constrains given by `proportion_checked` and `max_reviewers`. In this case, and depending on the inputs, it is possible that no perfect solution will exist, meaning that some reviewers may be allocated a higher proportion of articles than anticipated. For this reason it is often worth setting `quiet = FALSE` when running `allocate_effort` with variation in reviewer effort, to allow assessment of the results.

Value

Invisibly returns a data.frame giving one column per reviewer plus an extra column of proportions. The reviewer columns are binary and show which proportions apply to each person or combination of people.

See Also

`distribute_tasks` for how to use the output of `allocate_effort` to split a dataset.

Examples

```
# import some data
file_location <- system.file(
  "extdata",
  "avian_ecology_bibliography.ris",
  package = "revtools")
x <- read_bibliography(file_location)
# simple case - split evenly among 4 reviewers
result <- allocate_effort(4, quiet = TRUE)
# more complex - specify names and amount of overlap
result <- allocate_effort(
  reviewers = c("john", "paul", "george", "ringo"),
  proportion_checked = 0.2,
  max_reviewers = 3,
  quiet = TRUE
  )
# most complex - specify uneven effort among reviewers (experimental)
result <- allocate_effort(
  reviewers = 4,
  effort = c(0.9, 0.7, 0.5, 0.3),
  max_reviewers = 3,
  quiet = TRUE
  )
```

avian_ecology_bibliography *Bibliographic data from 20 papers on avian ecology*

Description

This dataset lists basic information (title, authors, keywords etc.) for 20 scientific articles on avian ecology, stored in .ris format.

Format

A list of length 20, containing lists of named attributes for each article.

Source

Originally downloaded from Scopus.

Examples

```
file_location <- system.file(
  "extdata",
  "avian_ecology_bibliography.ris",
  package = "revtools")
x <- read_bibliography(file_location)
summary(x)
```

bibliography-class *Description of class 'bibliography'*

Description

An S3 class designed to store data from common bibliographic formats in a standard way

Details

Class 'bibliography' is a nested list format; each object is a list containing multiple references, where each reference is a list with information on author, journal etc. Each entry has a unique name that is preserved in the 'label' column if the user subsequently calls `as.data.frame`.

See Also

`read_bibliography`, `write_bibliography`

bibliography-methods *Methods for class ’bibliography’*

Description

This is a small number of standard methods for interacting with class ’bibliography’. More may be added later.

Usage

```
as.bibliography(x, ...)
## S3 method for class 'bibliography'
as.data.frame(x, ...)
## S3 method for class 'bibliography'
x[n]
## S3 method for class 'bibliography'
c(...)
## S3 method for class 'bibliography'
print(x, n, ...)
## S3 method for class 'bibliography'
summary(object, ...)
```

Arguments

x	An object of class ’bibliography’
object	An object of class ’bibliography’
n	Number of items to select/print
...	Any further information

Examples

```
# import some data
file_location <- system.file(
  "extdata",
  "avian_ecology_bibliography.ris",
  package = "revtools")
x <- read_bibliography(file_location, return_df = FALSE)

# basic descriptions
summary(x)
print(x)
x[1]

# append two bibliography objects
y <- c(x[1:5], x[2:4])

# conversion to and from data.frame
y <- as.data.frame(x)
x_new <- as.bibliography(y)
```

`distribute_tasks` *Divide a set of articles among two or more reviewers*

Description

A common task in systematic review is to divide a dataset of articles located by a search (typically involving >1 databases) and distributing them amongst a team of reviewers for screening. This function takes a dataset divides it among the specified number of reviewers, returning the resulting data.frames either in a list to the workspace, or (by default) as a set of .csv files in the specified directory. The resulting files can be passed to any of the screening functions provided by revtools, i.e. `screen_titles`, `screen_abstracts`, or `screen_topics`.

Usage

```
distribute_tasks(data, reviewers, write_csv = TRUE,
    file_name = "reviewer.csv", return_data = FALSE, ...)
```

Arguments

`data`	a vector of strings
`reviewers`	Either a `data.frame` as returned by `allocate_effort`, an integer giving the number of reviewers, or a vector of strings giving reviewer names.
`write_csv`	Logical - should the function write a set of csv files (1 per reviewer)? Defaults to TRUE
`file_name`	a file path & name showing where .csv files should be saved. Ignored if `write_csv` is FALSE. Defaults to 'reviewer_[name].csv'.
`return_data`	Logical - should a list be (invisibly) returned, in which each entry is the data sent to a single reviewer? Defaults to FALSE.
`...`	Further arguments passed to `allocate_effort`

Details

The dataset is allocated each author in the proportion of articles specified by `allocate_effort`, with the identity of articles passed to reviewer being chosen by `rnorm`. As a result, this function is very sensitive to the inputs provided to `allocate_effort`, so it is often worth running that function first and checking the results to be certain that effort is being distributed in a way that you are happy with.

Value

Invisibly returns a list of data.frames, each with same columns as `data` but containing only a subset of rows.

See Also

`allocate_effort` for a detailed description of how the division among reviewers is accomplished.

Examples

```
# import some data
file_location <- system.file(
  "extdata",
  "avian_ecology_bibliography.ris",
  package = "revtools")
x <- read_bibliography(file_location)
result <- distribute_tasks(x, 4, write_csv = FALSE) # split evenly among 4 reviewers
```

extract_unique_references

Create a de-duplicated data.frame

Description

Take a data.frame of bibliographic information showing potential duplicates (as returned by find_duplicates), and return a data.frame of unique references.

Usage

```
extract_unique_references(x, matches)
```

Arguments

x	a data.frame to be subsetted
matches	either a vector of matches, e.g. as returned from find_duplicates, or a column name (specified as a number or a string) from x showing where matches are stored

Value

a subsetted data.frame containing one entry for each group identified in matches.

Note

This function creates a simplified version of x, by extracting the reference from each group of ’identical’ references that contains the most text. It is assumed that this is the most ’complete’ record of those available in the dataset. This function does not merge data from multiple ’identical’ records due to the potential for mis-matching that this approach would create.

See Also

find_duplicates for duplicate identification; screen_duplicates for an interactive alternative to duplicate removal.

Examples

```
# import data
file_location <- system.file(
  "extdata",
  "avian_ecology_bibliography.ris",
  package = "revtools"
)
x <- read_bibliography(file_location)

# generate duplicated references (for example purposes)
x_duplicated <- rbind(x, x[1:5,])

# locate and extract unique references
x_check <- find_duplicates(x_duplicated)
x_unique <- extract_unique_references(x_duplicated, matches = x_check)
```

`find_duplicates` *Locate duplicated information within a data.frame*

Description

Identify potential duplicates within a `data.frame`.

Usage

```
find_duplicates(data, match_variable, group_variables,
  match_function, method, threshold,
  to_lower = FALSE, remove_punctuation = FALSE)
```

Arguments

`data` a `data.frame` containing data to be matched

`match_variable` a length-1 integer or string listing the column in which duplicates should be sought. Defaults to doi where available, followed by title. If neither are found the function will fail.

`group_variables` an optional vector listing the columns to use as grouping variables; that is, categories withing which duplicates should be sought (see ’note’). Optionally NULL to compare all entries against one another.

`match_function` a function to calculate dissimilarity between strings. Defaults to "exact" if doi’s are available or "stringdist" otherwise.

`method` the required ’method’ option that corresponds with `match_function`. Defaults to NULL if `match_function` is "exact", "osa" for match_function == "stringdist", or "fuzz_m_ratio" for match_function == "fuzzdist".

`threshold` an upper limit above which similar articles are not recognized as duplicates. Defaults to 5 for stringdist and 0.1 for fuzzdist. Ignored if `match_function` == "exact".

to_lower	logical: should text be made lower case prior to searching? Defaults to FALSE.
remove_punctuation	logical: should punctuation be removed prior to searching? Defaults to FALSE.

Value

an integer vector, in which entries with the same integer have been selected as duplicates by the selected algorithm.

Note

find_duplicates runs a while loop. It starts by checking the first entry of data against every other entry for potential duplicates. If any matches are found, those entries are excluded from consideration. The loop then continues until all entries have been checked. In order to work, this function requires the data and match_variable arguments be specified. The remaining arguments affects how duplicates are identified, and can also strongly influence the speed of the outcome.

The argument group_variables specifies variables that contain supplementary information that can reduce the number of entries that need to be searched. For example, you might want to only match article titles if they occur within the same journal, or in the same year. The more variables you specify, the fewer pairs of matches that have to be tested to locate duplicates, greatly increasing the speed of the algorithm. Conversely, if no variables are specified, then each entry is checked against every other entry that has yet to be excluded from the dataset. This is fine for small datasets, but massively increases computation time for large datasets.

Missing values are handled differently. Entries that are NA for match_variable are always labelled as unique values, and are not checked for duplicates against the rest of the dataset. However, entries of group_variables that are NA are included in every comparison.

find_duplicates contains three 'built-in' methods for string matching. "stringdist" calls the function of the same name from the package stringdist; ditto for "fuzzdist" which is in revtools, but based on the Python library fuzzywuzzy. "exact" simply searches for exact matches. In principle you could call any function for string matching, so long as it accepts the arguments a, b and method (see documentation on stringdist for details), and returns a measure of distance (i.e. not similarity).

Finally, to_lower and remove_punctuation specify whether to transform the text prior to searching for duplicates.

See Also

screen_duplicates and extract_unique_references for manual and automated screening (respectively) of results from this function.

Examples

```
# import data
file_location <- system.file(
  "extdata",
  "avian_ecology_bibliography.ris",
  package = "revtools")
x <- read_bibliography(file_location)

# generate then locate some 'fake' duplicates
```

```
x_duplicated <- rbind(x, x[1:5,])
x_check <- find_duplicates(x_duplicated)
# returns a vector of potential matches
```

format_citation *Format a citation*

Description

takes an object of class data.frame or bibliography and returns a formatted citation.

Usage

```
format_citation(data, details = TRUE, abstract = FALSE,
  add_html = FALSE, line_breaks = FALSE, ...)
```

Arguments

data	An object of class data.frame or bibliography.
details	Logical: Should identifying information such as author names & journal titles be displayed? Defaults to TRUE.
abstract	Logical: Should the abstract be shown (if available)? Defaults to FALSE.
add_html	Logical: Should the journal title be italicized using html codes? Defaults to FALSE.
line_breaks	Either logical, stating whether line breaks should be added, or numeric stating how many characters should separate consecutive line breaks. Defaults to FALSE.
...	any other arguments.

Value

a string of length == length(x), containing formatted citations.

Examples

```
file_location <- system.file(
  "extdata",
  "avian_ecology_bibliography.ris",
  package = "revtools")
x <- read_bibliography(file_location, return_df = FALSE)
format_citation(x[[1]])
format_citation(as.data.frame(x)[1, ]) # same result
```

fuzz_functions *Functions for fuzzy string matching*

Description

Duplicate of functions from the Python library 'fuzzywuzzy' (https://github.com/seatgeek/fuzzywuzzy). These functions have been recoded from scratch based on the description given here. For consistency with `stringdist`, however, these functions are computed as distances rather than similarities; i.e. low values signify similar strings.

Usage

```
fuzzdist(a, b, method)
fuzz_m_ratio(a, b)
fuzz_partial_ratio(a, b)
fuzz_token_sort_ratio(a, b)
fuzz_token_set_ratio(a, b)
```

Arguments

`a`	a string
`b`	a vector containing one or more strings
`method`	a function to be called by 'fuzzdist'

Value

A score of same length as y, giving the proportional dissimilarity between x and y.

Note

`fuzz_m_ratio` is a measure of the number of letters that match between two strings. It is calculated as one minus two times the number of matched characters, divided by the number of characters in both strings.

`fuzz_partial_ratio` calculates the extent to which one string is a subset of the other. If one string is a perfect subset, then this will be zero.

`fuzz_token_sort_ratio` sorts the words in both strings into alphabetical order, and checks their similarity using `fuzz_m_ratio`.

`fuzz_token_set_ratio` is similar to `fuzz_token_sort_ratio`, but compares both sorted strings to each other, and to a third group made of words common to both strings. It then returns the maximum value of `fuzz_m_ratio` from these comparisons.

`fuzzdist` is a wrapper function, for compatability with stringdist.

Examples

```
fuzz_m_ratio("NEW YORK METS", "NEW YORK MEATS")
fuzz_partial_ratio(
  "YANKEES",
  c("NEW YORK YANKEES", "something else", "YNAKEES")
)
fuzz_token_sort_ratio("New York Mets vs Atlanta Braves", "Atlanta Braves vs New York Melts")
fuzz_token_set_ratio(
  a = "mariners vs angels other words",
  b = c("los angeles angels of anaheim at seattle mariners", "angeles angels of anaheim ")
)
```

`make_dtm` *Construct a document-term matrix (DTM)*

Description

Takes bibliographic data and converts it to a DTM for passing to topic models.

Usage

```
make_dtm(x, stop_words, min_freq = 0.01, max_freq = 0.85,
  bigram_check = TRUE, bigram_quantile = 0.8,
  retain_empty_rows = FALSE)
```

Arguments

`x` a vector or `data.frame` containing text

`stop_words` optional vector of strings, listing terms to be removed from the DTM prior to analysis. Defaults to `revwords()`.

`min_freq` minimum proportion of entries that a term must be found in to be retained in the analysis. Defaults to 0.01.

`max_freq` maximum proportion of entries that a term must be found in to be retained in the analysis. Defaults to 0.85.

`bigram_check` logical: should ngrams be searched for?

`bigram_quantile`

what quantile of ngrams should be retained. Defaults to 0.8; i.e. the 80th percentile of bigram frequencies after removing all bigrams with frequencies <=2.

`retain_empty_rows`

logical: should the function return an object with the same length as the input string (TRUE), or remove rows that contain no text after filtering rare & common words etc? (FALSE, the default). The latter is the default because this is required by `run_topic_model`.

Details

This is primarily intended to be called internally by `screen_topics`, but is made available for users to generate their own topic models with the same properties as those in revtools.

This function uses some standard tools like stemming, converting words to lower case, and removal of numbers or punctuation. It also replaces stemmed words with the shortest version of all terms that share the same stem, which doesn't affect the calculations, but makes the resulting analyses easier to interpret. It doesn't use part-of-speech tagging.

Words that occur in 2 entries or fewer are always removed by `make_dtm`, so values of `min_freq` that result in a threshold below this will not affect the result. Arguments to `max_freq` are passed as is. Similarly words consisting of three letters or fewer are removed.

If `retain_empty_rows` is FALSE (the default) and the object returned is named `z`, then `as.numeric(z$dimnames$Docs)` provides an index to which entries have been retained from the input vector (`x`).

Value

An object of class `simple_triplet_matrix`, listing the terms (columns) present in each row or string.

See Also

`run_topic_model`, `screen_topics`

Examples

```
# import some data
file_location <- system.file(
  "extdata",
  "avian_ecology_bibliography.ris",
  package = "revtools")
x <- read_bibliography(file_location)

# construct a document-term matrix
# note: this can take a long time to run for large datasets
x_dtm <- make_dtm(x$title)

# to return a matrix where length(output) == nrow(x)
x_dtm <- make_dtm(x$title, retain_empty_rows = TRUE)
x_dtm <- as.matrix(x_dtm) # to convert to class 'matrix'
dim(x_dtm) # 20 articles, 10 words
```

`merge_columns` *rbind two or more data frames with different columns*

Description

Take two data.frames with arbitraily different numbers and names of columns, and rbind them together.

Usage

```
merge_columns(x, y)
```

Arguments

`x`	A data.frame, or a list containing multiple data.frames.
`y`	An optional second data.frame. Only used if x is not a list.

Value

An object of class 'data.frame', containing data from all input data frames. That is, all unique columns are preserved, while rows that are missing data for a given column are assigned NA.

Examples

```
# import some data
file_location <- system.file(
  "extdata",
  "avian_ecology_bibliography.ris",
  package = "revtools")
x <- read_bibliography(file_location)
y <- x[, 1:3]
z <- merge_columns(x, y)
# or equivalently
z <- merge_columns(list(x, y))
```

read_bibliography *Import bibliographic data*

Description

Import standard formats from academic search engines and referencing software.

Usage

```
read_bibliography(filename, return_df = TRUE)
```

Arguments

`filename`	A vector or list containing paths to one or more bibliographic files.
`return_df`	logical; should the object returned be a data.frame? Defaults to TRUE. If FALSE, returns an object of class `bibliography`.

Details

This function aims to import bibliographic data from a range of formats in a consistent manner.

If the user provides a number of paths in a vector or list, then files are imported in sequence and joined into a single data.frame (or a list if return_df = FALSE) using merge_columns. If return_df = TRUE (the default) then an extra column called 'file_name' is appended to the resulting data.frame to show the file in which each entry originated.

If the file is in .csv format, then read_bibliography will import the file using read.csv, with three changes. First, it ensures that the first column contains an index (i.e. a unique value for each row), and creates one if it is absent. Second, it converts column names to lower case and switches all delimiters to underscores. Third, it ensures that author names are delimited by 'and' for consistency with format_citation.

If the file is of any type other than .csv, read_bibliography auto-detects document formatting, first by detecting whether the document is ris-like or bib-like, and then running an appropriate import function depending on the result. In the case of ris-like files (including files from 'Endnote' & 'Web of Science'), this involves attempting to detect both the delimiter between successive entries, and the means of separating tag labels (e.g. 'AU', 'TI') from their information content. Attempts have been made to ensure consistency with .ris, .bib, medline (.nbib) or web of science (.ciw) formats. Except for .csv, file extensions are not used to determine file type, and are ignored except to locate the file.

If the imported file is in a ris-like format, then the object returned by read_bibliography will have different headings from the source document. This feature attempts to ensure consistency across file types. Tag substitutions are made using a lookup table, which can be viewed by calling tag_lookup. Unrecognized tags are grouped in the resulting bibliography object under the heading 'further_info'.

Value

Returns an object of class data.frame if return_df is TRUE; otherwise an object of class bibliography.

See Also

bibliography-class, tag_lookup

Examples

```
file_location <- system.file(
  "extdata",
  "avian_ecology_bibliography.ris",
  package = "revtools")
x <- read_bibliography(file_location)
class(x) # = data.frame
x <- read_bibliography(file_location, return_df = FALSE)
class(x) # = bibliography
summary(x)
```

revwords *Load a set of stopwords*

Description

Generates a list of stopwords, consisting of all the terms given by `tm::stopwords`, plus some extra terms (mainly words that designate numbers).

Usage

```
revwords()
```

Value

A vector of stopwords in English.

Note

This is primarily an internal function, but may be useful in other contexts.

Examples

```
# import some data
file_location <- system.file(
  "extdata",
  "avian_ecology_bibliography.ris",
  package = "revtools")
x <- read_bibliography(file_location)

# construct a document-term matrix
x_dtm <- make_dtm(x$title,
  stop_words = revwords())
# Note that make_dtm calls revwords by default, so this is technically redundant
```

run_topic_model *Calculate a topic model*

Description

Calculate a topic model using Latent Dirichlet Allocation (`LDA`) or Correlated Topic Models (`CTM`), using the `topicmodels` package.

Usage

```
run_topic_model(dtm, type = "lda", n_topics = 5, iterations = 2000)
```

Arguments

dtm	a Document Term Matrix (DTM)
type	string specififying the type of model to run. Either 'lda' (the default) or 'ctm'.
n_topics	Number of topics to calculate
iterations	The number of iterations. Only relevant for LDA.

Value

A topic model with the specified parameters.

Note

This is a basic wrapper function designed to allow consistent specification of model parameters within shiny apps.

See Also

make_dtm for constructing data to pass to this function; screen_topics for interactive visualisation of topic model results.

Examples

```
# import data
file_location <- system.file(
  "extdata",
  "avian_ecology_bibliography.ris",
  package = "revtools"
)
x <- read_bibliography(file_location)

# run a topic model based on these data
# note: the following lines can take a very long time to run, especially for large datasets!
x_dtm <- make_dtm(x$title)
## Not run: x_lda <- run_topic_model(x_dtm, "lda", 5, 5000)
```

screen_abstracts *Shiny app for screening articles by their abstracts*

Description

This is a simple app for displaying bibliographic data one entry at a time, and manually selecting or excluding them. Articles can be ordered by a user-specified column, or or in one of three automated ways: as in the input dataset, alphabetically by title, or in random order (the default).

Usage

```
screen_abstracts(x, max_file_size)
```

Arguments

`x` An (optional) object of class `data.frame` or `bibliography` to open in the browser. If empty, the app will launch with no data. Data can be added within the app via the 'import' button.

`max_file_size` Optional argument to set the maximum file size (in MB) that the app will accept.

Value

This function launches a Shiny app in the users' default browser, allowing the user to select or exclude individual articles.

See Also

`screen_titles` for screening articles in groups rather than individually; `screen_topics` to view articles as a point cloud.

Examples

```
# to run the app and upload data interactively
## Not run: screen_abstracts()
# or to specify data from the workspace
file_location <- system.file(
  "extdata",
  "avian_ecology_bibliography.ris",
  package = "revtools")
x <- read_bibliography(file_location)
# to run the app using these data:
## Not run: screen_abstracts(x)
# or to run the app & save results to the workspace:
## Not run: result <- screen_abstracts(x)
```

`screen_duplicates` *Shiny app for locating and excluding duplicated entries in a dataset*

Description

This is a simple app for calculating, displaying and screening potential duplicates in bibliographic data

Usage

```
screen_duplicates(x, max_file_size)
```

Arguments

`x` An (optional) object of class `data.frame` or `bibliography` to open in the browser. If empty, the app will launch with no data. Data can be added within the app via the 'import' button.

`max_file_size` Optional argument to set the maximum file size (in MB) that the app will accept.

Details

This app is effectively a wrapper for find_duplicates, with the added option to manually screen pairs of duplicates to check the results. Consequently, this is a more reliable method than extract_unique_references of dealing with the duplicates identified by find_duplicates, and for testing whether that function has returned sensible results for a given dataset.

Value

This function launches a Shiny app in the users' default browser, allowing the user to customize their parameters for duplicate detection, and visualise the results.

See Also

screen_titles or screen_abstracts for manual screening of individual articles.

Examples

```
# to run the app and upload data interactively
## Not run: screen_duplicates()
# or to specify data from the workspace
file_location <- system.file(
  "extdata",
  "avian_ecology_bibliography.ris",
  package = "revtools")
x <- read_bibliography(file_location)
# to run the app using these data:
## Not run: screen_duplicates(x)
# or to run the app & save results to the workspace:
## Not run: result <- screen_duplicates(x)
```

---

screen_titles *Shiny app for screening articles by their titles*

Description

This is a simple app for displaying the titles in a bibliographic dataset in small groups, and manually selecting or excluding them. Articles can be ordered as in the input dataset, alphabetically by title, or in random order (the default).

Usage

```
screen_titles(x, max_file_size)
```

Arguments

x	An (optional) object of class data.frame or bibliography to open in the browser. If empty, the app will launch with no data. Data can be added within the app via the 'import' button.
max_file_size	Optional argument to set the maximum file size (in MB) that the app will accept.

Value

This function launches a Shiny app in the users' default browser, allowing the user to select or exclude articles.

See Also

`screen_abstracts` for screening articles one at a time rather than in groups; `screen_topics` to view articles as a point cloud.

Examples

```
# to run the app and upload data interactively
## Not run: screen_titles()
# or to specify data from the workspace
file_location <- system.file(
  "extdata",
  "avian_ecology_bibliography.ris",
  package = "revtools")
x <- read_bibliography(file_location)
# to run the app using these data:
## Not run: screen_titles(x)
# or to run the app & save results to the workspace:
## Not run: result <- screen_titles(x)
```

`screen_topics` *Shiny app for screening bibliographies using topic models*

Description

Screening is usually achieved by manually sorting titles or abstracts one at a time. `screen_topics` offers an alternative by allowing the user to group data by any column in the input dataset, and running a topic model on the resulting data. This allows a great deal of flexibility to locate patterns in journals, years, or authors, rather than just articles. Data points can be selected or excluded individually, or by topic.

Usage

```
screen_topics(x, remove_words, max_file_size)
```

Arguments

`x`	An (optional) object of class `data.frame` or `bibliography` to open in the browser. If empty, the app will launch with no data. Data can be added within the app via the 'import' button.
`remove_words`	Optional vector of words to be removed from consideration by the Topic Model. If none are given, `screen_topics` will use `revwords`. Note that this vector will be converted to lower case before processing, so the algorithm is not case sensitive.
`max_file_size`	Optional argument to set the maximum file size (in MB) that the app will accept.

Details

The display space is divided into three parts. From left to right, these are the sidebar; the plot window; and the selection panel.

The sidebar shows a series of drop-down menus that can be used to customize or recalculate the central plot. It can be hidden when not in use. Note that the default settings for LDA (5 topics, 10,000 iterations) prioritize speed over reliability - higher numbers of iterations will give more reliable results.

The plot window shows an ordination of article weights calculated using LDA, with articles colored by their highest-weighted topic. Hovering over a point shows the title and abstract below the plot; clicking allows selection or deselection of that article (and optionally displays co-authorship data). Selecting a region of the plot and clicking zooms on the selected region; double-clicking without selecting a region returns the plot to its full extent.

The selection panel gives information on progress in selecting/deselecting articles. It also contains windows for displaying topic-level infromation and article abstracts. All boxes in this panel can be minimized when not required.

Ordinations are calculated using LDA (library `"topicmodels"`) and are displayed using `shiny` and `plotly`.

When you have finished viewing/screening, you can export information to a .csv or .rda file (saved to the working directory) using the 'Save' tab.

Note that "`start_review_window`" is the earlier form of this function; this has been deprecated and will be removed from future versions of `revtools`.

Value

This function launches a Shiny app in the users' default browser.

See Also

`screen_titles` or `screen_abstracts` for manual screening; `screen_topics_progress-class` for saving and restoring progress in `screen_topics`.

Examples

```
# to run the app and upload data interactively
## Not run: screen_topics()
# or to specify data from the workspace
file_location <- system.file(
  "extdata",
  "avian_ecology_bibliography.ris",
  package = "revtools")
x <- read_bibliography(file_location)
# to run the app using these data:
## Not run: screen_topics(x)
# or to run the app & save results to the workspace:
## Not run: result <- screen_topics(x)
```

`screen_topics_progress-class` *Description of class 'screen_topics_progress'*

Description

`screen_topics_progress` is an S3 class designed to store data from `screen_topics`, allowing the user to re-load a previously calculated topic model. If you just want to save your decisions on article inclusion/exclusion, along with your notes, then this is probably overkill as that information can simply be exported as a .csv file.

slots

Class 'screen_topics_progress' has seven slots containing the following information:

- **raw** duplicate of data passed to `screen_topics`
- **stopwords** vector of words excluded from the dtm
- **columns** vector of column names in the original dataset
- **grouped** a data.frame showing grouped data as specified by the user
- **dtm** document-term matrix, created by `make_dtm`
- **model** most recent topic model, created by `run_topic_model`
- **plot_ready** data needed for the main plot (coordinates etc.)

`screen_topics_progress-methods` *Methods for class 'screen_topics_progress'*

Description

Tools to display useful information on class `screen_topics_progress`.

Usage

```
## S3 method for class 'screen_topics_progress'
summary(object, ...)
```

Arguments

`object`	An object of class 'screen_topics_progress'
`...`	Any further information

Value

Prints useful information to the workspace.

Note

Class `screen_topics_progress` is a format for exporting large quantities of data during reviews. It is typically stored within a .rds file in the working directory. When re-imported to R using `readRDS`, this file will contain an object of class `screen_topics_progress`.

`tag_lookup` *Lookup table for ris tags*

Description

ris-like bibliographic data files contain codes that describe their contents, such as 'AU' in place of 'author'. This function provides lookup tables for 'ris' tags of different kinds

Usage

```
tag_lookup(type = "ris")
```

Arguments

`type`	Which lookup table should be returned? Accepted values are 'ris', 'ris_write' or 'medline'

Details

Primarily an internal function to support `read_bibliography` and `write_bibliography`. Tag substitutions for PubMed/Medline fields are taken directly from the NIH (available here). Substitutions for other ris-like formats are based on common examples, but are much less consistently documented.

Value

a `data.frame` containing the original tag (column 'ris'), and the full-word substitution for that tag (column 'bib'). For type 'ris', there is also an added 'order' column showing the order those tags should be displayed in. 'ris_write' is a version of 'ris' with only one ris tag per bib tag.

See Also

`bibliography-class`, `read_bibliography`

Examples

```
tag_lookup("ris") # standard ris format
tag_lookup("medline") # PubMed files
```

write_bibliography *Export imported bibliographic data as .bib or .ris formats*

Description

Basic function to export bibliographic information for use in other programs. Work in progress. Very little error checking or advanced formatting in this version

Usage

```
write_bibliography(x, filename, format = "ris")
```

Arguments

x	An object of class 'bibliography', such as imported using read_bibliography
filename	Name of the exported file. Should ideally match 'format', but this is not enforced
format	Format of the exported file. Should be either "ris" (default) or "bib"

Value

exports results as a .ris or .bib file.

Examples

```
file_location <- system.file(
  "extdata",
  "avian_ecology_bibliography.ris",
  package = "revtools")
x <- read_bibliography(file_location, return_df = FALSE)

# export a subset of entries as a new file
write_bibliography(x[1:5],
  filename = paste0(tempdir(), "/x_out.ris"),
  format = "ris")
```

Index