Package 'lingmatch'

January 26, 2022

Type Package

Title Linguistic Matching and Accommodation

Version 1.0.3

Author Micah Iserman

Maintainer Micah Iserman <micah.iserman@gmail.com>

Description Measure similarity between texts. Offers a variety of processing
tools and similarity metrics to facilitate flexible representation of texts and matching.
Implements forms of Language Style Matching (Ireland & Pennebaker, 2010) <doi:10.1037/a0020386>
and Latent Semantic Analysis (Landauer & Dumais, 1997) <doi:10.1037/0033-295X.104.2.211>.

URL https://github.com/miserman/lingmatch

BugReports https://github.com/miserman/lingmatch/issues

Depends R (>= 3.5), methods, Matrix

Imports Rcpp, RcppParallel

License GPL (>= 2)

RoxygenNote 7.1.2

Suggests knitr, rmarkdown, splot, testthat (>= 2.1.0)

LinkingTo Rcpp, RcppParallel

SystemRequirements C++11

NeedsCompilation yes

Repository CRAN

Date/Publication 2022-01-25 23:52:46 UTC

R topics documented:

`download.dict` *Download Dictionaries*

Description

Downloads the specified dictionaries from osf.io/y6g5b.

Usage

```
download.dict(dict = "lusi", check.md5 = TRUE, mode = "wb",
  dir = getOption("lingmatch.dict.dir"))
```

Arguments

`dict`	One or more names of dictionaries to download, or `'all'` for all available. See osf.io/y6g5b/wiki for more information, and a list of available dictionaries.
`check.md5`	Logical; if `TRUE` (default), retrieves the MD5 checksum from OSF, and compares it with that calculated from the downloaded file to check its integrity.
`mode`	A character specifying the file write mode; default is 'wb'. See `download.file`.
`dir`	Directory in which to save the dictionary; default is `getOption('lingmatch.dict.dir')`. This must be specified, or the option must be set – use `lma_initdirs` to initialize a directory.

Value

Path to the downloaded dictionary, or a list of such if multiple were downloaded.

See Also

Other Dictionary functions: `lma_patcat()`, `lma_termcat()`, `read.dic()`, `select.dict()`

Examples

```
## Not run:

download.dict('lusi', dir = '~/Dictionaries')


## End(Not run)
```

`download.lspace` *Download Latent Semantic Spaces*

Description

Downloads the specified semantic space from osf.io/489he.

Usage

```
download.lspace(space = "100k_lsa", include.terms = TRUE,
  decompress = TRUE, check.md5 = TRUE, mode = "wb",
  dir = getOption("lingmatch.lspace.dir"))
```

Arguments

`space`	Name of one or more spaces you want to download, or `'all'` for all available. `'100k_lsa'` is the default, and some other common options might be `'google'`, `'facebook'`, or `'glove'`. See osf.io/489he/wiki for more information, and a full list of spaces.
`include.terms`	Logical; if `FALSE`, only the `.dat.bz2` file is downloaded (which only has numeric vectors).
`decompress`	Logical; if `TRUE` (default), decompresses the downloaded file with the `bunzip2` system command assuming it is available (as indicated by `Sys.which('bunzip2')`).
`check.md5`	Logical; if `TRUE` (default), retrieves the MD5 checksum from OSF, and compares it with that calculated from the downloaded file to check its integrity.
`mode`	A character specifying the file write mode; default is 'wb'. See `download.file`.
`dir`	Directory in which to save the space. Specify this here, or set the lspace directory option (e.g., `options(lingmatch.lspace.dir = '~/Latent Semantic Spaces')`), or use `lma_initdirs` to initialize a directory.

Value

A character vector with paths to the [1] data and [2] term files.

See Also

Other Latent Semantic Space functions: `lma_lspace()`, `select.lspace()`, `standardize.lspace()`

Examples

```
## Not run:

download.lspace('glove_crawl', dir = '~/Latent Semantic Spaces')


## End(Not run)
```

`lingmatch` *Linguistic Matching and Accommodation*

Description

Offers a variety of methods to assess linguistic matching or accommodation, where *matching* is general similarity (sometimes called *homophily*), and *accommodation* is some form of conditional similarity (accounting for some base-rate or precedent; sometimes called *alignment*).

Usage

```
lingmatch(input = NULL, comp = mean, data = NULL, group = NULL, ...,
  comp.data = NULL, comp.group = NULL, order = NULL, drop = FALSE,
  all.levels = FALSE, type = "lsm")
```

Arguments

`input` Texts to be compared; a vector, document-term matrix (dtm; with terms as column names), or path to a file (.txt or .csv, with texts separated by one or more lines/rows).

`comp` Defines the comparison to be made:

- If a **function**, this will be applied to `input` within each group (overall if there is no group; i.e., `apply(input,2,comp)`; e.g., `comp = mean` would compare each text to the mean profile of its group).
- If a **character** with a length of 1 and no spaces:
  - If it partially matches one of `lsm_profiles`'s rownames, that row will be used as the comparison.
  - If it partially matches `'auto'`, the highest correlating `lsm_profiles` row will be used.
  - If it partially matches `'pairwise'`, each text will be compared to one another.
  - If it partially matches `'sequential'`, the last variable in `group` will be treated as a speaker ID (see the Grouping and Comparisons section).
- If a **character vector**, this will be processed in the same way as `input`.

- If a **vector**, either (a) logical or factor-like (having n levels < length) and of the same length as `nrow(input)`, or (b) numeric or logical of length less than `nrow(input)`, this will be used to select a subset of `input` (e.g., `comp = 1:10` would treat the first 10 rows of `input` as the comparison; `comp = type == 'prompt'` would make a logical vector identifying prompts, assuming "type" was the name of a column in `data`, or a variable in the global environment, and the value "prompt" marked the prompts).
- If a **matrix-like object** (having multiple rows and columns), or a named vector, this will be treated as a sort of dtm, assuming there are common (column) names between `input` and `comp` (e.g., if you had prompt and response texts that were already processed separately).

`data` A matrix-like object as a reference for column names, if variables are referred to in other arguments (e.g., `lingmatch(text,data = data)` would be the same as `lingmatch(data$text)`.

`group` A logical or factor-like vector the same length as `NROW(input)`, used to defined groups.

`...` Passes arguments to `lma_dtm`, `lma_weight`, `lma_termcat`, and/or `lma_lspace` (depending on `input` and `comp`), and `lma_simets`.

`comp.data` A matrix-like object as a source for `comp` variables.

`comp.group` The column name of the grouping variable(s) in `comp.data`; if `group` contains references to column names, and `comp.group` is not specified, `group` variables will be looked for in `comp.data`.

`order` A numeric vector the same length as `nrow(input)` indicating the order of the texts and grouping variables when the type of comparison is sequential. Only necessary if the texts are not already ordered as desired.

`drop` logical; if `FALSE`, columns with a sum of 0 are retained.

`all.levels` logical; if `FALSE`, multiple groups are combined. See the Grouping and Comparisons section.

`type` A character at least partially matching 'lsm' or 'lsa'; applies default settings aligning with the standard calculations of each type:

LSM	`lingmatch(text, weight = 'freq', dict = lma_dict(1:9), metric = 'canberra')`
LSA	`lingmatch(text, weight = 'tfidf', space = '100k_lsa', metric = 'cosine')`

Details

There are a great many points of decision in the assessment of linguistic similarity and/or accommodation, partly inherited from the great many point of decision inherent in the numerical representation of language. Two general types of matching are implemented here as sets of defaults: Language/Linguistic Style Matching (LSM; Niederhoffer & Pennebaker, 2002; Ireland & Pennebaker, 2010), and Latent Semantic Analysis/Similarity (LSA; Landauer & Dumais, 1997; Babcock, Ta, & Ickes, 2014). See the `type` argument for specifics.

Value

A list with processed components of the input, information about the comparison, and results of the comparison:

- `dtm`: A sparse matrix; the raw count-dtm, or a version of the original input if it is more processed.
- `processed`: A matrix-like object; a processed version of the input (e.g., weighted and categorized).
- `comp.type`: A string describing the comparison if applicable.
- `comp`: A vector or matrix-like object; the comparison data if applicable.
- `group`: A string describing the group if applicable.
- `sim`: Result of `lma_simets`.

Grouping and Comparisons

Defining groups and comparisons can sometimes be a bit complicated, and requires dataset specific knowledge, so it can't always (readily) be done automatically. Variables entered in the `group` argument are treated differently depending on their position and other arguments:

Splitting By default, groups are treated as if they define separate chunks of data in which comparisons should be calculated. Functions used to calculated comparisons, and pairwise comparisons are performed separately in each of these groups. For example, if you wanted to compare each text with the mean of all texts in its condition, a `group` variable could identify and split by condition. Given multiple grouping variables, calculations will either be done in each split (if `all.levels = TRUE`; applied in sequence so that groups become smaller and smaller), or once after all splits are made (if `all.levels = FALSE`). This makes for 'one to many' comparisons with either calculated or preexisting standards (i.e., the profile of the current data, or a precalculated profile, respectively).

Comparison ID When comparison data is identified in `comp`, groups are assumed to apply to both `input` and `comp` (either both in `data`, or separately between `data` and `comp.data`, in which case `comp.group` may be needed if the same grouping variable have different names between `data` and `comp.data`). In this case, multiple grouping variables are combined into a single factor assumed to uniquely identify a comparison. This makes for 'one to many' comparisons with specific texts (as in the case of manipulated prompts or text-based conditions).

Speaker ID If `comp` matches `'sequential'`, the last grouping variable entered is assumed to identify something like speakers (i.e., a factor with two or more levels and multiple observations per level). In this case, the data are assumed to be ordered (or ordered once sorted by `order` if specified). Any additional grouping variables before the last are treated as splitting groups. This can set up for probabilistic accommodation metrics. At the moment, when sequential comparisons are made within groups, similarity scores between speakers are averaged, resulting in mean matching between speakers within the group.

References

Babcock, M. J., Ta, V. P., & Ickes, W. (2014). Latent semantic similarity and language style matching in initial dyadic interactions. *Journal of Language and Social Psychology, 33*, 78-88.

Ireland, M. E., & Pennebaker, J. W. (2010). Language style matching in writing: synchrony in essays, correspondence, and poetry. *Journal of Personality and Social Psychology, 99*, 549.

Landauer, T. K., & Dumais, S. T. (1997). A solution to Plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological Review, 104*, 211.

Niederhoffer, K. G., & Pennebaker, J. W. (2002). Linguistic style matching in social interaction. *Journal of Language and Social Psychology, 21*, 337-360.

See Also

For a general text processing function, see `lma_process`.

Examples

```
# compare single strings
lingmatch('Compare this sentence.', 'With this other sentence.')

# compare each entry in a character vector with...
texts = c(
  'One bit of text as an entry...',
  'Maybe multiple sentences in an entry. Maybe essays or posts or a book.',
  'Could be lines or a column from a read-in file...'
)

## one another
lingmatch(texts)

## the first
lingmatch(texts, 1)

## the next
lingmatch(texts, 'seq')

## the set average
lingmatch(texts, mean)

## other entries in a group
lingmatch(texts, group = c('a', 'a', 'b'))

## one another, without stop words
lingmatch(texts, exclude = 'function')

## a standard average (based on function words)
lingmatch(texts, 'auto', dict = lma_dict(1:9))
```

lma_dict *English Function Word Category and Special Character Lists*

Description

Returns a list of function words based on the Linguistic Inquiry and Word Count 2015 dictionary (in terms of category names – words were selected independently), or a list of special characters and patterns.

Usage

```
lma_dict(..., as.regex = TRUE, as.function = FALSE)
```

Arguments

...	Numbers or letters corresponding to category names: ppron, ipron, article, adverb, conj, prep, auxverb, negate, quant, interrog, number, interjection, or special.
as.regex	Logical: if FALSE, lists are returned without regular expression.
as.function	Logical or a function: if specified and as.regex is TRUE, the selected dictionary will be collapsed to a regex string (terms separated by |), and a function for matching characters to that string will be returned. The regex string is passed to the matching function (grepl by default) as a 'pattern' argument, with the first argument of the returned function being passed as an 'x' argument. See examples.

Value

A list with a vector of terms for each category, or (when as.function = TRUE) a function which accepts an initial "terms" argument (a character vector), and any additional arguments determined by function entered as as.function (grepl by default).

Note

The special category is not returned unless specifically requested. It is a list of regular expression strings attempting to capture special things like ellipses and emojis, or sets of special characters (those outside of the Basic Latin range; [^\u0020-\u007F]), which can be used for character conversions. If special is part of the returned list, as.regex is set to TRUE.

The special list is always used by both lma_dtm and lma_termcat. When creating a dtm, special is used to clean the original input (so that, by default, the punctuation involved in ellipses and emojis are treated as different – as ellipses and emojis rather than as periods and parens and colons and such). When categorizing a dtm, the input dictionary is passed by the special lists to be sure the terms in the dtm match up with the dictionary (so, for example, ": (" would be replaced with "repfrown" in both the text and dictionary).

See Also

To score texts with these categories, use lma_termcat.

Examples

```
# return the full dictionary (excluding special)
lma_dict()

# return the standard 7 category lsm categories
lma_dict(1:7)

# return just a few categories without regular expression
lma_dict(neg, ppron, aux, as.regex=FALSE)

# return special specifically
lma_dict(special)

# returning a function
is.ppron = lma_dict(ppron, as.function = TRUE)
is.ppron(c('i', 'am', 'you', 'were'))

in.lsmcat = lma_dict(1:7, as.function = TRUE)
in.lsmcat(c('a', 'frog', 'for', 'me'))

## use as a stopword filter
is.stopword = lma_dict(as.function = TRUE)
dtm = lma_dtm('Most of these words might not be all that relevant.')
dtm[, !is.stopword(colnames(dtm))]

## use to replace special characters
clean = lma_dict(special, as.function = gsub)
clean(c(
  "\u201Ccurly quotes\u201D", 'na\u00EFve', 'typographer\u2019s apostrophe',
  'en\u2013dash', 'em\u2014dash'
))
```

---

lma_dtm *Document-Term Matrix Creation*

Description

Creates a document-term matrix (dtm) from a set of texts.

Usage

```
lma_dtm(text, exclude = NULL, context = NULL, replace.special = TRUE,
  numbers = FALSE, punct = FALSE, urls = TRUE, emojis = FALSE,
  to.lower = TRUE, word.break = " +", dc.min = 0, dc.max = Inf,
  sparse = TRUE, tokens.only = FALSE)
```

Arguments

text	Texts to be processed. This can be a vector (such as a column in a data frame) or list.
exclude	A character vector of words to be excluded. If exclude is a single string matching 'function', lma_dict(1:9) will be used.
context	A character vector used to reformat text based on look- ahead/behind. For example, you might attempt to disambiguate *like* by reformatting certain *like*s (e.g., context = c('(i) like*','(you) like*','(do) like'), where words in parentheses are the context for the target word, and asterisks denote partial matching). This would be converted to regular expression (i.e., '(?<= i) like\\b') which, if matched, would be replaced with a coded version of the word (e.g., "Hey, i like that!" would become "Hey, i i-like that!"). This would probably only be useful for categorization, where a dictionary would only include one or another version of a word (e.g., the LIWC 2015 dictionary does something like this with *like*, and LIWC 2007 did something like this with *kind (of)*, both to try and clean up the posemo category).
replace.special	Logical: if TRUE (default), special characters are replaced with regular equivalents using the lma_dict special function.
numbers	Logical: if TRUE, numbers are preserved.
punct	Logical: if TRUE, punctuation is preserved.
urls	Logical: if FALSE, attempts to replace all urls with "repurl".
emojis	Logical: if TRUE, attempts to replace emojis (e.g., ":(" would be replaced with "repfrown").
to.lower	Logical: if FALSE, words with different capitalization are treated as different terms.
word.break	A regular expression string determining the way words are split. Default is ' +' which breaks words at one or more blank spaces. You may also like to break by dashes or slashes ('[/-]+'), depending on the text.
dc.min	Numeric: excludes terms appearing in the set number or fewer documents. Default is 0 (no limit).
dc.max	Numeric: excludes terms appearing in the set number or more. Default is Inf (no limit).
sparse	Logical: if FALSE, a regular dense matrix is returned.
tokens.only	Logical: if TRUE, returns a list rather than a matrix, with these entries:

tokens	A vector of indices with terms as names.
frequencies	A vector of counts with terms as names.
WC	A vector of term counts for each document.
indices	A list with a vector of token indices for each document.

Value

A sparse matrix (or regular matrix if `sparse = FALSE`), with a row per `text`, and column per term, or a list if `tokens.only = TRUE`. Includes an attribute with options (`opts`), and attributes with word count (`WC`) and column sums (`colsums`) if `tokens.only = FALSE`.

Note

This is a relatively simple way to make a dtm. To calculate the (more or less) standard forms of LSM and LSS, a somewhat raw dtm should be fine, because both processes essentially use dictionaries (obviating stemming) and weighting or categorization (largely obviating 'stop word' removal). The exact effect of additional processing will depend on the dictionary/semantic space and weighting scheme used (particularly for LSA). This function also does some processing which may matter if you plan on categorizing with categories that have terms with look- ahead/behind assertions (like LIWC dictionaries). Otherwise, other methods may be faster, more memory efficient, and/or more featureful.

Examples

```
text = c(
  "Why, hello there! How are you this evening?",
  "I am well, thank you for your inquiry!",
  "You are a most good at social interactions person!",
  "Why, thank you! You're not all bad yourself!"
)

lma_dtm(text)
```

---

lma_initdirs *Initialize Directories for Dictionaries and Latent Semantic Spaces*

Description

Creates directories for dictionaries and latent semantic spaces if needed, sets them as the `lingmatch.dict.dir` and `lingmatch.lspace.dir` options if they are not already set, and creates links to them in their expected locations (`'~/Dictionaries'` and `'~/Latent Semantic Spaces'`) by default if applicable.

Usage

```
lma_initdirs(base = "", dict = "Dictionaries",
  lspace = "Latent Semantic Spaces", link = TRUE)
```

Arguments

`base`	Path to a directory in which to create the `dict` and `lspace` subdirectories.
`dict`	Path to the dictionaries directory relative to `base`.
`lspace`	Path to the latent semantic spaces directory relative to `base`.

link — Logical; if TRUE (default), the full dict and/or lspace paths exist (potentially after being created), and they are not '~/Dictionaries' or '~/Latent Semantic Spaces' respectively, junctions (Windows) or symbolic links will be created: ~/Dictionaries <<===>> dict and ~/Latent Semantic Spaces <<===>> lspace.

Value

Paths to the [1] dictionaries and [2] latent semantic space directories, or a single path if only dict or lspace is specified.

Examples

```
## Not run:

# set up the expected dictionary and latent semantic space directories
lma_initdirs('~')

# set up directories elsewhere, and links to the expected locations
lma_initdirs('d:')

# point options and create links to preexisting directories
lma_initdirs('~/NLP_Resources', 'Dicts', 'Dicts/Embeddings')

# create just a dictionaries directory and set the
# lingmatch.dict.dir option without creating a link
lma_initdirs(dict = 'z:/external_dictionaries', link = FALSE)


## End(Not run)
```

---

lma_lspace — *Latent Semantic Space (Embeddings) Operations*

Description

Map a document-term matrix onto a latent semantic space, extract terms from a latent semantic space (if dtm is a character vector, or map.space = FALSE), or perform a singular value decomposition of a document-term matrix (if dtm is a matrix and space is missing).

Usage

```
lma_lspace(dtm = "", space, map.space = TRUE, fill.missing = FALSE,
  term.map = NULL, dim.cutoff = 0.5, keep.dim = FALSE,
  use.scan = FALSE, dir = getOption("lingmatch.lspace.dir"))
```

Arguments

dtm	A matrix with terms as column names, or a character vector of terms to be extracted from a specified space. If this is of length 1 and space is missing, it will be treated as space.
space	A matrix with terms as rownames. If missing, this will be the right singular vectors of a singular value decomposition of dtm. If a character, a file matching the character will be searched for in dir (e.g., space = 'google'). If a file is not found and the character matches one of the available spaces, you will be given the option to download it, as handled by download.lspace. If dtm is missing, the entire space will be loaded and returned.
map.space	Logical: if FALSE, the original vectors of space for terms found in dtm are returned. Otherwise dtm %*% space is returned, excluding uncommon columns of dtm and rows of space.
fill.missing	Logical: if TRUE and terms are being extracted from a space, includes terms not found in the space as rows of 0s, such that the returned matrix will have a row for every requested term.
term.map	A matrix with space as a column name, terms as row names, and indices of the terms in the given space as values, or a numeric vector of indices with terms as names, or a character vector or terms corresponding to rows of the space. This is used instead of reading in an "_terms.txt" file corresponding to a space entered as a character (the name of a space file).
dim.cutoff	If a space is calculated, this will be used to decide on the number of dimensions to be retained: cumsum(d) / sum(d) < dim.cutoff, where d is a vector of singular values of dtm (i.e., svd(dtm)$d). The default is .5; lower cutoffs result in fewer dimensions.
keep.dim	Logical: if TRUE, and a space is being calculated from the input, a matrix in the same dimensions as dtm is returned. Otherwise, a matrix with terms as rows and dimensions as columns is returned.
use.scan	Logical: if TRUE, reads in the rows of space with scan.
dir	Path to a folder containing spaces. Set a session default with options(lingmatch.lspace.dir = 'desired/path').

Value

A matrix or sparse matrix with either (a) a row per term and column per latent dimension (a latent space, either calculated from the input, or retrieved when map.space = FALSE), (b) a row per document and column per latent dimension (when a dtm is mapped to a space), or (c) a row per document and column per term (when a space is calculated and keep.dim = TRUE).

Note

A traditional latent semantic space is a selection of right singular vectors from the singular value decomposition of a dtm (svd(dtm)$v[,1:k], where k is the selected number of dimensions, decided here by dim.cutoff).

Mapping a new dtm into a latent semantic space consists of multiplying common terms: dtm[,ct] %*% space[ct,], where ct = colnames(dtm)[colnames(dtm) %in% rownames(space)] – the

terms common between the dtm and the space. This results in a matrix with documents as rows, and dimensions as columns, replacing terms.

See Also

Other Latent Semantic Space functions: download.lspace(), select.lspace(), standardize.lspace()

Examples

```
text = c(
  paste(
    "Hey, I like kittens. I think all kinds of cats really are just the",
    "best pet ever."
  ),
  paste(
    "Oh year? Well I really like cars. All the wheels and the turbos...",
    "I think that's the best ever."
  ),
  paste(
    "You know what? Poo on you. Cats, dogs, rabbits -- you know, living",
    "creatures... to think you'd care about anything else!"
  ),
  paste(
    "You can stick to your opinion. You can be wrong if you want. You know",
    "what life's about? Supercharging, diesel guzzling, exhaust spewing,",
    "piston moving ignitions."
  )
)

dtm = lma_dtm(text)

# calculate a latent semantic space from the example text
lss = lma_lspace(dtm)

# show that document similarities between the truncated and full space are the same
spaces = list(
  full = lma_lspace(dtm, keep.dim = TRUE),
  truncated = lma_lspace(dtm, lss)
)
sapply(spaces, lma_simets, metric = 'cosine')

## Not run:

# specify a directory containing spaces,
# or where you would like to download spaces
space_dir = '~/Latent Semantic Spaces'

# map to a pretrained space
ddm = lma_lspace(dtm, '100k', dir = space_dir)

# load the matching subset of the space
# without mapping
lss_100k_part = lma_lspace(colnames(dtm), '100k', dir = space_dir)
```

```
## or
lss_100k_part = lma_lspace(dtm, '100k', map.space = FALSE, dir = space_dir)

# load the full space
lss_100k = lma_lspace('100k', dir = space_dir)

## or
lss_100k = lma_lspace(space = '100k', dir = space_dir)


## End(Not run)
```

`lma_meta` *Calculate Text-Based Metastatistics*

Description

Calculate simple descriptive statistics from text.

Usage

```
lma_meta(text)
```

Arguments

`text`	A character vector of texts.

Value

A data.frame:

- `characters`: Total number of characters.
- `syllables`: Total number of syllables, as estimated by split length of `'a+[eu]*|e+a*|i+|o+[ui]*|u+|y+[aeiou]*'` - 1.
- `words`: Total number of words (raw word count).
- `unique_words`: Number of unique words (binary word count).
- `clauses`: Number of clauses, as marked by commas, colons, semicolons, dashes, or brackets within sentences.
- `sentences`: Number of sentences, as marked by periods, question marks, exclamation points, or new line characters.
- `words_per_clause`: Average number of words per clause.
- `words_per_sentence`: Average number of words per sentence.
- `sixltr`: Number of words 6 or more characters long.
- `characters_per_word`: Average number of characters per word (`characters` / `words`).
- `syllables_per_word`: Average number of syllables per word (`syllables` / `words`).

- type_token_ratio: Ratio of unique to total words: unique_words / words.
- reading_grade: Flesch-Kincaid grade level: .39 * words / sentences + 11.8 * syllables / words - 15.59.
- numbers: Number of terms starting with numbers.
- punct: Number of terms starting with non-alphanumeric characters.
- periods: Number of periods.
- commas: Number of commas.
- qmarks: Number of question marks.
- exclams: Number of exclamation points.
- quotes: Number of quotation marks (single and double).
- apostrophes: Number of apostrophes, defined as any modified letter apostrophe, or backtick or single straight or curly quote surrounded by letters.
- brackets: Number of bracketing characters (including parentheses, and square, curly, and angle brackets).
- orgmarks: Number of characters used for organization or structuring (including dashes, forward slashes, colons, and semicolons).

Examples

```
text = c(
  succinct = "It is here.",
  verbose = "Hear me now. I shall tell you about it. It is here. Do you hear?",
  couched = "I might be wrong, but it seems to me that it might be here.",
  bigwords = "Object located thither.",
  excited = "It's there! It's there! It's there!",
  drippy = "It's 'there', right? Not 'here'? 'there'? Are you Sure?",
  struggly = "It's here -- in that place where it is. Like... the 1st place (here)."
)
lma_meta(text)
```

---

lma_patcat *Categorize Texts*

Description

Categorize raw texts using a pattern-based dictionary.

Usage

```
lma_patcat(text, dict = NULL, pattern.weights = "weight",
  pattern.categories = "category", bias = NULL, to.lower = TRUE,
  return.dtm = FALSE, drop.zeros = FALSE, exclusive = TRUE,
  boundary = NULL, fixed = TRUE, globtoregex = FALSE,
  name.map = c(intname = "_intercept", term = "term"),
  dir = getOption("lingmatch.dict.dir"))
```

Arguments

text	A vector of text to be categorized. Texts are padded by 2 spaces, and potentially lowercased.
dict	At least a vector of terms (patterns), usually a matrix-like object with columns for terms, categories, and weights.
pattern.weights	A vector of weights corresponding to terms in dict, or the column name of weights found in dict.
pattern.categories	A vector of category names corresponding to terms in dict, or the column name of category names found in dict.
bias	A constant to add to each category after weighting and summing. Can be a vector with names corresponding to the unique values in dict[,category], but is usually extracted from dict based on the intercept included in each category (defined by name.map['intname']).
to.lower	Logical indicating whether text should be converted to lowercase before processing.
return.dtm	Logical; if TRUE, only a document-term matrix will be returned, rather than the weighted, summed, and biased category values.
drop.zeros	logical; if TRUE, categories or terms with no matches will be removed.
exclusive	Logical; if FALSE, each dictionary term is searched for in the original text. Otherwise (by default), terms are sorted by length (with longer terms being searched for first), and matches are removed from the text (avoiding subsequent matches to matched patterns).
boundary	A string to add to the beginning and end of each dictionary term. If TRUE, boundary will be set to ' ', avoiding pattern matches within words. By default, dictionary terms are left as entered.
fixed	Logical; if FALSE, patterns are treated as regular expressions.
globtoregex	Logical; if TRUE, initial and terminal asterisks are replaced with \\b\\w* and \\w*\\b respectively. This will also set fixed to FALSE unless fixed is specified.
name.map	A named character vector: • intname: term identifying category biases within the term list; defaults to '_intercept' • term: name of the column containing terms in dict; defaults to 'term' Missing names are added, so names can be specified positional (e.g., c('_int', 'terms')), or only some can be specified by name (e.g., c(term = 'patterns')), leaving the rest default.
dir	Path to a folder in which to look for dict if it is the name of a file to be passed to read.dic.

Value

A matrix with a row per text and columns per dictionary category, or (when return.dtm = TRUE) a sparse matrix with a row per text and column per term. Includes a WC attribute with original word counts, and a categories attribute with row indices associated with each category if return.dtm = TRUE.

See Also

For applying term-based dictionaries (to a document-term matrix) see lma_termcat.

Other Dictionary functions: download.dict(), lma_termcat(), read.dic(), select.dict()

Examples

```
# example text
text = c(
  paste(
    "Oh, what youth was! What I had and gave away.",
    "What I took and spent and saw. What I lost. And now? Ruin."
  ),
  paste(
    "God, are you so bored?! You just want what's gone from us all?",
    "I miss the you that was too. I love that you."
  ),
  paste(
    "Tomorrow! Tomorrow--nay, even tonight--you wait, as I am about to change.",
    "Soon I will off to revert. Please wait."
  )
)

# make a document-term matrix with pre-specified terms only
lma_patcat(text, c('bored?!', 'i lo', '. '), return.dtm = TRUE)

# get counts of sets of letter
lma_patcat(text, list(c('a', 'b', 'c'), c('d', 'e', 'f')))

# same thing with regular expressions
lma_patcat(text, list('[abc]', '[def]'), fixed = FALSE)

# match only words
lma_patcat(text, list('i'), boundary = TRUE)

# match only words, ignoring punctuation
lma_patcat(
  text, c('you', 'tomorrow', 'was'), fixed = FALSE,
  boundary = '\\b', return.dtm = TRUE
)

## Not run:

# read in the temporal orientation lexicon from the World Well-Being Project
tempori = read.csv(
  'https://wwbp.org/downloads/public_data/temporalOrientationLexicon.csv'
)

lma_patcat(text, tempori)

# or use the standardized version
tempori_std = read.dic('wwbp_prospection', dir = '~/Dictionaries')
```

```
lma_patcat(text, tempori_std)

## get scores on the same scale by adjusting the standardized values
tempori_std[, -1] = tempori_std[, -1] / 100 *
  select.dict('wwbp_prospection')$selected[, 'original_max']

lma_patcat(text, tempori_std)[, unique(tempori$category)]


## End(Not run)
```

`lma_process` *Process Text*

Description

A wrapper to other pre-processing functions, potentially from `read.segments`, to `lma_dtm` or `lma_patcat`, to `lma_weight`, then `lma_termcat` or `lma_lspace`, and optionally including `lma_meta` output.

Usage

```
lma_process(input = NULL, ..., meta = TRUE)
```

Arguments

`input`	A vector of text, or path to a text file or folder.
`...`	arguments to be passed to `lma_dtm`, `lma_patcat`, `lma_weight`, `lma_termcat`, and/or `lma_lspace`. All arguments must be named.
`meta`	Logical; if `FALSE`, metastatistics are not included. Only applies when raw text is available. If included, meta categories are added as the last columns, with names starting with "meta_".

Value

A matrix with texts represented by rows, and features in columns, unless there are multiple rows per output (e.g., when a latent semantic space is applied without terms being mapped) in which case only the special output is returned (e.g., a matrix with terms as rows and latent dimensions in columns).

See Also

If you just want to compare texts, see the `lingmatch` function.

Examples

```
# starting with some texts in a vector
texts = c(
 'Firstly, I would like to say, and with all due respect...',
 'Please, proceed. I hope you feel you can speak freely...',
 "Oh, of course, I just hope to be clear, and not cause offense...",
 "Oh, no, don't monitor yourself on my account..."
)

# by default, term counts and metastatistics are returned
lma_process(texts)

# add dictionary and percent arguments for standard dictionary-based results
lma_process(texts, dict = lma_dict(), percent = TRUE)

# add space and weight arguments for standard word-centroid vectors
lma_process(texts, space = lma_lspace(texts), weight = 'tfidf')
```

`lma_simets` *Similarity Calculations*

Description

Enter a numerical matrix, set of vectors, or set of matrices to calculate similarity per vector.

Usage

```
lma_simets(a, b = NULL, metric = NULL, group = NULL, lag = 0,
  agg = TRUE, agg.mean = TRUE, pairwise = TRUE, symmetrical = FALSE,
  mean = FALSE, return.list = FALSE)
```

Arguments

`a`	A vector or matrix. If a vector, `b` must also be provided. If a matrix and `b` is missing, each row will be compared. If a matrix and `b` is not missing, each row will be compared with `b` or each row of `b`.
`b`	A vector or matrix to be compared with `a` or rows of `a`.
`metric`	A character or vector of characters at least partially matching one of the available metric names (or 'all' to explicitly include all metrics), or a number or vector of numbers indicating the metric by index: • `jaccard`: `sum(a & b) / sum(a | b)` • `euclidean`: `1 / (1 + sqrt(sum((a - b) ^ 2)))` • `canberra`: `mean(1 - abs(a - b) / (a + b))` • `cosine`: `sum(a * b) / sqrt(sum(a ^ 2 * sum(b ^ 2)))` • `pearson`: `(mean(a * b) - (mean(a) * mean(b))) / sqrt(mean(a ^ 2) - mean(a) ^ 2) / sqrt(mean(b ^ 2) - mean(b) ^ 2)`

group	If b is missing and a has multiple rows, this will be used to make comparisons between rows of a, as modified by agg and agg.mean.
lag	Amount to adjust the b index; either rows if b has multiple rows (e.g., for lag = 1, a[1,] is compared with b[2,]), or values otherwise (e.g., for lag = 1, a[1] is compared with b[2]). If b is not supplied, b is a copy of a, resulting in lagged self-comparisons or autocorrelations.
agg	Logical: if FALSE, only the boundary rows between groups will be compared, see example.
agg.mean	Logical: if FALSE aggregated rows are summed instead of averaged.
pairwise	Logical: if FALSE and a and b are matrices with the same number of rows, only paired rows are compared. Otherwise (and if only a is supplied), all pairwise comparisons are made.
symmetrical	Logical: if TRUE and pairwise comparisons between a rows were made, the results in the lower triangle are copied to the upper triangle.
mean	Logical: if TRUE, a single mean for each metric is returned per row of a.
return.list	Logical: if TRUE, a list-like object will always be returned, with an entry for each metric, even when only one metric is requested.

Details

Use setThreadOptions to change parallelization options; e.g., run RcppParallel::setThreadOptions(4) before a call to lma_simets to set the number of CPU threads to 4.

Value

Output varies based on the dimensions of a and b:

- **Out:** A vector with a value per metric.
  In: Only when a and b are both vectors.
- **Out:** A vector with a value per row.
  In: Any time a single value is expected per row: a or b is a vector, a and b are matrices with the same number of rows and pairwise = FALSE, a group is specified, or mean = TRUE, and only one metric is requested.
- **Out:** A data.frame with a column per metric.
  In: When multiple metrics are requested in the previous case.
- **Out:** A sparse matrix with a metric attribute with the metric name.
  In: Pairwise comparisons within an a matrix or between an a and b matrix, when only 1 metric is requested.
- **Out:** A list with a sparse matrix per metric.
  In: When multiple metrics are requested in the previous case.

Examples

```
text = c(
  'words of speaker A', 'more words from speaker A',
  'words from speaker B', 'more words from speaker B'
```

```
)
(dtm = lma_dtm(text))

# compare each entry
lma_simets(dtm)

# compare each entry with the mean of all entries
lma_simets(dtm, colMeans(dtm))

# compare by group (corresponding to speakers and turns in this case)
speaker = c('A', 'A', 'B', 'B')

## by default, consecutive rows from the same group are averaged:
lma_simets(dtm, group = speaker)

## with agg = FALSE, only the rows at the boundary between
## groups (rows 2 and 3 in this case) are used:
lma_simets(dtm, group = speaker, agg = FALSE)
```

`lma_termcat` *Document-Term Matrix Categorization*

Description

Reduces the dimensions of a document-term matrix by dictionary-based categorization.

Usage

```
lma_termcat(dtm, dict, term.weights = NULL, bias = NULL,
  bias.name = "_intercept", escape = TRUE, partial = FALSE,
  glob = TRUE, term.filter = NULL, term.break = 20000,
  to.lower = FALSE, dir = getOption("lingmatch.dict.dir"))
```

Arguments

`dtm`	A matrix with terms as column names.
`dict`	The name of a provided dictionary (osf.io/y6g5b/wiki) or of a file found in `dir`, or a `list` object with named character vectors as word lists, or the path to a file to be read in by `read.dic`.
`term.weights`	A `list` object with named numeric vectors lining up with the character vectors in `dict`, used to weight the terms in each `dict` vector. If a category in `dict` is not specified in `term.weights`, or the `dict` and `term.weights` vectors aren't the same length, the weights for that category will be 1.
`bias`	A list or named vector specifying a constant to add to the named category. If a term matching `bias.name` is included in a category, it's associated `weight` will be used as the `bias` for that category.
`bias.name`	A character specifying a term to be used as a category bias; default is `'_intercept'`.

escape	Logical indicating whether the terms in dict should not be treated as plain text (including asterisk wild cards). If TRUE, regular expression related characters are escaped. Set to TRUE if you get PCRE compilation errors.
partial	Logical; if TRUE terms are partially matched (not padded by ^ and $).
glob	Logical; if TRUE (default), will convert initial and terminal asterisks to partial matches.
term.filter	A regular expression string used to format the text of each term (passed to gsub). For example, if terms are part-of-speech tagged (e.g., 'a_DT'), '_.*' would remove the tag.
term.break	If a category has more than term.break characters, it will be processed in chunks. Reduce from 20000 if you get a PCRE compilation error.
to.lower	Logical; if TRUE will lowercase dictionary terms. Otherwise, dictionary terms will be converted to match the terms if they are single-cased. Set to FALSE to always keep dictionary terms as entered.
dir	Path to a folder in which to look for dict; will look in '~/Dictionaries' by default. Set a session default with options(lingmatch.dict.dir = 'desired/path').

Value

A matrix with a row per dtm row and columns per dictionary category, and a WC attribute with original word counts.

See Also

For applying pattern-based dictionaries (to raw text) see lma_patcat.

Other Dictionary functions: download.dict(), lma_patcat(), read.dic(), select.dict()

Examples

```
## Not run:

# Score texts with the NRC Affect Intensity Lexicon

dict = readLines('https://saifmohammad.com/WebDocs/NRC-AffectIntensity-Lexicon.txt')
dict = read.table(
  text = dict[-seq_len(grep('term\tscore', dict, fixed = TRUE)[[1]])],
  col.names = c('term', 'weight', 'category')
)

text = c(
  angry = paste(
    'We are outraged by their hateful brutality,',
    'and by the way they terrorize us with their hatred.'
  ),
  fearful = paste(
    'The horrific torture of that terrorist was tantamount',
    'to the terrorism of terrorists.'
  ),
```

```
  joyous = 'I am jubilant to be celebrating the bliss of this happiest happiness.',
  sad = paste(
    'They are nearly suicidal in their mourning after',
    'the tragic and heartbreaking holocaust.'
  )
)

emotion_scores = lma_termcat(text, dict)
if(require('splot')) splot(emotion_scores ~ names(text), leg = 'out')

## or use the standardized version (which includes more categories)

emotion_scores = lma_termcat(text, 'nrc_eil', dir = '~/Dictionaries')
emotion_scores = emotion_scores[, c('anger', 'fear', 'joy', 'sadness')]
if(require('splot')) splot(emotion_scores ~ names(text), leg = 'out')

## End(Not run)
```

`lma_weight` *Document-Term Matrix Weighting*

Description

Weight a document-term matrix.

Usage

```
lma_weight(dtm, weight = "count", normalize = TRUE, wc.complete = TRUE,
  log.base = 10, alpha = 1, pois.x = 1L, doc.only = FALSE,
  percent = FALSE)
```

Arguments

`dtm`	A matrix with words as column names.

`weight`	A string referring at least partially to one (or a combination; see note) of the available weighting methods:
	Term weights (applied uniquely to each cell)
	• `binary`
	`(dtm > 0) * 1`
	Convert frequencies to 1s and 0s; remove differences in frequencies.
	• `log`
	`log(dtm + 1,log.base)`
	Log of frequencies.
	• `sqrt`
	`sqrt(dtm)`
	Square root of frequencies.

- `count`
  `dtm`
  Unaltered; sometimes called term frequencies (tf).
- `amplify`
  `dtm ^ alpha`
  Amplify difference in frequencies.

Document weights (applied by column)

- `dflog`
  `log(colSums(dtm > 0),log.base)`
  Log of binary term sum.
- `entropy`
  `1 -rowSums(x * log(x + 1,log.base) / log(ncol(x),log.base), na.rm = TRUE)`
  Where `x = t(dtm) / colSums(dtm > 0)`; entropy of term-conditional term distribution.
- `ppois`
  `1 -ppois(pois.x, colSums(dtm) / nrow(dtm))`
  Poisson-predicted term distribution.
- `dpois`
  `1 -dpois(pois.x,colSums(dtm) / nrow(dtm))`
  Poisson-predicted term density.
- `dfmlog`
  `log(diag(dtm[max.col(t(dtm)),]),log.base)`
  Log of maximum term frequency.
- `dfmax`
  `diag(dtm[max.col(t(dtm)),])`
  Maximum term frequency.
- `df`
  `colSums(dtm > 0)`
  Sum of binary term occurrence across documents.
- `idf`
  `log(nrow(dtm) / colSums(dtm > 0),log.base)`
  Inverse document frequency.
- `ridf`
  `idf -log(dpois,log.base)`
  Residual inverse document frequency.
- `normal`
  `sqrt(1 / colSums(dtm ^ 2))`
  Normalized document frequency.

Alternatively, `'pmi'` or `'ppmi'` will apply a pointwise mutual information weighting scheme (with `'ppmi'` setting negative values to 0).

`normalize` Logical: if `FALSE`, the dtm is not divided by document word-count before being weighted.

`wc.complete` If the dtm was made with `lma_dtm` (has a `'WC'` attribute), word counts for frequencies can be based on the raw count (default; `wc.complete = TRUE`). If

	wc.complete = FALSE, or the dtm does not have a 'WC' attribute, rowSums(dtm) is used as word count.
log.base	The base of logs, applied to any weight using log. Default is 10.
alpha	A scaling factor applied to document frequency as part of pointwise mutual information weighting, or amplify's power (dtm ^ alpha, which defaults to 1.1).
pois.x	integer; quantile or probability of the poisson distribution (dpois(pois.x,colSums(x, na.rm = TRUE) / nrow(x))).
doc.only	Logical: if TRUE, only document weights are returned (a single value for each term).
percent	Logical; if TRUE, frequencies are multiplied by 100.

Value

A weighted version of dtm, with a type attribute added (attr(dtm,'type')).

Note

Term weights works to adjust differences in counts within documents, with differences meaning increasingly more from binary to log to sqrt to count to amplify.

Document weights work to treat words differently based on their between-document or overall frequency. When term frequencies are constant, dpois, idf, ridf, and normal give less common words increasingly more weight, and dfmax, dfmlog, ppois, df, dflog, and entropy give less common words increasingly less weight.

weight can either be a vector with two characters, corresponding to term weight and document weight (e.g., c('count','idf')), or it can be a string with term and document weights separated by any of :*_/; ,- (e.g., 'count-idf'). 'tf' is also acceptable for 'count', and 'tfidf' will be parsed as c('count','idf'), though this is a special case.

For weight, term or document weights can be entered individually; term weights alone will not apply any document weight, and document weights alone will apply a 'count' term weight (unless doc.only = TRUE, in which case a term-named vector of document weights is returned instead of a weighted dtm).

Examples

```
# visualize term and document weights

## term weights
term_weights = c('binary', 'log', 'sqrt', 'count', 'amplify')
Weighted = sapply(term_weights, function(w) lma_weight(1:20, w, FALSE))
if(require(splot)) splot(Weighted ~ 1:20, labx = 'Raw Count', lines = 'co')

## document weights
doc_weights = c('df', 'dflog', 'dfmax', 'dfmlog', 'idf', 'ridf',
  'normal', 'dpois', 'ppois', 'entropy')
weight_range = function(w, value = 1){
  m = diag(20)
  m[upper.tri(m, TRUE)] = if(is.numeric(value)) value else unlist(lapply(
    1:20, function(v) rep(if(value == 'inverted') 21 - v else v, v)
```

```
  ))
  lma_weight(m, w, FALSE, doc.only = TRUE)
}

if(require(splot)){
  category = rep(c('df', 'idf', 'normal', 'poisson', 'entropy'), c(4, 2, 1, 2, 1))
  op = list(
    laby = 'Relative (Scaled) Weight', labx = 'Document Frequency',
    leg = 'outside', colorby = list(quote(category), grade = TRUE),
    lines = 'connected', mv.scale = TRUE, note = FALSE
  )
  splot(
    sapply(doc_weights, weight_range) ~ 1:20,
    options = op, title = 'Same Term, Varying Document Frequencies',
    sud = 'All term frequencies are 1.'
  )
  splot(
    sapply(doc_weights, weight_range, value = 'sequence') ~ 1:20,
    options = op, title = 'Term as Document Frequencies',
    sud = 'Non-zero terms are the number of non-zero terms.'
  )
  splot(
    sapply(doc_weights, weight_range, value = 'inverted') ~ 1:20,
    options = op, title = 'Term Opposite of Document Frequencies',
    sud = 'Non-zero terms are the number of zero terms + 1.'
  )
}
```

`read.dic` *Read/Write Dictionary Files*

Description

Read in or write dictionary files in Comma-Separated Values (.csv; weighted) or Linguistic Inquiry and Word Count (.dic; non-weighted) format.

Usage

```
read.dic(path, cats, type = "asis", as.weighted = FALSE,
  dir = getOption("lingmatch.dict.dir"), ..., term.name = "term",
  category.name = "category", raw = FALSE)

write.dic(dict, filename, type = "asis", as.weighted = FALSE, save = TRUE)
```

Arguments

`path`	Path to a file, a name corresponding to a file in `getOption('lingmatch.dict.dir')` (or `'~/Dictionaries'`) or one of the dictionaries available at osf.io/y6g5b, a matrix-like object to be categorized, or a list to be formatted.

cats — A character vector of category names to be returned. All categories are returned by default.

type — A character indicating whether and how terms should be altered. Unspecified or matching 'asis' leaves terms as they are. Other options change wildcards to regular expressions: 'pattern' ('^[poi]') replaces initial asterisks with '\\b\\w*', and terminal asterisks with '\\w*\\b', to match terms within raw text; for anything else, terms are padded with ^ and $, then those bounding marks are removed when an asterisk is present, to match tokenized terms.

as.weighted — Logical; if TRUE, prevents weighted dictionaries from being converted to unweighted versions, or converts unweighted dictionaries to a binary weighted version – a data.frame with a "term" column of unique terms, and a column for each category.

dir — Path to a folder containing dictionaries, or where you would like dictionaries to be downloaded; passed to select.dict and/or download.dict.

... — Passes arguments to readLines.

term.name, category.name — Strings identifying column names in path containing terms and categories respectively.

raw — Logical or a character. As logical, indicates if path should be treated as a raw dictionary (as might be read in from a .dic file). As a character, replaces path as if it were read in from a file.

dict — A list with a named entry of terms for each category, or a data.frame with terms in one column, and categories or weights in the rest.

filename — The name of the file to be saved.

save — Logical: if FALSE, does not write a file.

Value

read.dic: A list (unweighted) with an entry for each category containing character vectors of terms, or a data.frame (weighted) with columns for terms (first, "term") and weights (all subsequent, with category labels as names).

write.dic: A version of the written dictionary – a raw character vector for unweighted dictionaries, or a data.frame for weighted dictionaries.

See Also

Other Dictionary functions: download.dict(), lma_patcat(), lma_termcat(), select.dict()

Examples

```
# make a small murder related dictionary
dict = list(
  kill = c('kill*', 'murd*', 'wound*', 'die*'),
  death = c('death*', 'dying', 'die*', 'kill*')
)

# convert it to a weighted format
```

```
(dict_weighted = read.dic(dict, as.weighted = TRUE))

# categorize it back
read.dic(dict_weighted)

# convert it to a string without writing to a file
(raw_dict = write.dic(dict, save = FALSE))

# parse it back in
read.dic(raw = raw_dict)

## Not run:

# save it as a .dic file
write.dic(dict, 'murder')

# read it back in as a list
read.dic('murder.dic')

# read in the Moral Foundations or LUSI dictionaries from urls
moral_dict = read.dic('https://osf.io/download/whjt2')
lusi_dict = read.dic('https://www.depts.ttu.edu/psy/lusi/files/lusi_dict.txt')

# save and read in a version of the General Inquirer dictionary
inquirer = read.dic('inquirer', dir = '~/Dictionaries')


## End(Not run)
```

`read.segments` *Read and Segment Multiple Texts*

Description

Split texts by word count or specific characters. Input texts directly, or read them in from files.

Usage

```
read.segments(path = ".", segment = NULL, ext = ".txt", subdir = FALSE,
  segment.size = -1, bysentence = FALSE, end_in_quotes = TRUE,
  preclean = FALSE, text = NULL)
```

Arguments

`path`	Path to a folder containing files, or a vector of paths to files. If no folders or files are recognized in `path`, it is treated as `text`.
`segment`	Specifies how the text of each file should be segmented. If a character, split at that character; '\n' by default. If a number, texts will be broken into that many segments, each with a roughly equal number of words.

ext	The extension of the files you want to read in. '.txt' by default.
subdir	Logical; if TRUE, files in folders in path will also be included.
segment.size	Logical; if specified, segment will be ignored, and texts will be broken into segments containing roughly segment.size number of words.
bysentence	Logical; if TRUE, and segment is a number or segment.size is specified, sentences will be kept together, rather than potentially being broken across segments.
end_in_quotes	Logical; if FALSE, sentence-ending marks (.?!) will not be considered when immediately followed by a quotation mark. For example, '"Word." Word.' would be considered one sentence.
preclean	Logical; if TRUE, text will be cleaned with lma_dict(special) before segmentation.
text	A character vector with text to be split, used in place of path. Each entry is treated as a file.

Value

A data.frame with columns for file names (input), segment number within file (segment), word count for each segment (WC), and the text of each segment (text).

Examples

```
# split preloaded text
read.segments('split this text into two segments', 2)

## Not run:

# read in all files from the package directory
texts = read.segments(path.package('lingmatch'), ext = '')
texts[, -4]

# segment .txt files in dir in a few ways:
dir = 'path/to/files'

## into 1 line segments
texts_lines = read.segments(dir)

## into 5 even segments each
texts_5segs = read.segments(dir, 5)

## into 50 word segments
texts_50words = read.segments(dir, segment.size = 50)

## into 1 sentence segments
texts_1sent = read.segments(dir, segment.size = 1, bysentence = TRUE)


## End(Not run)
```

`select.dict` *Select Dictionaries*

Description

Retrieve information and links to dictionaries (lexicons/word lists) available at osf.io/y6g5b.

Usage

```
select.dict(query = NULL, dir = getOption("lingmatch.dict.dir"),
  check.md5 = TRUE, mode = "wb")
```

Arguments

`query`	A character matching a dictionary name, or a set of keywords to search for in dictionary information.
`dir`	Path to a folder containing dictionaries, or where you want them to be saved. Will look in getOption('lingmatch.dict.dir') and '~/Dictionaries' by default.
`check.md5`	Logical; if `TRUE` (default), retrieves the MD5 checksum from OSF, and compares it with that calculated from the downloaded file to check its integrity.
`mode`	Passed to `download.file` when downloading files.

Value

A list with varying entries:

- `info`: The version of osf.io/kjqb8 stored internally; a `data.frame` with dictionary names as row names, and information about each dictionary in columns.
  Also described at osf.io/y6g5b/wiki/dict_variables, here `short` (corresponding to the file name [`{short}.(csv|dic)`] and wiki urls [`https://osf.io/y6g5b/wiki/{short}`]) is set as row names and removed:
  - `name`: Full name of the dictionary.
  - `description`: Description of the dictionary, relating to its purpose and development.
  - `note`: Notes about processing decisions that additionally alter the original.
  - `constructor`: How the dictionary was constructed:
    - `algorithm`: Terms were selected by some automated process, potentially learned from data or other resources.
    - `crowd`: Several individuals rated the terms, and in aggregate those ratings translate to categories and weights.
    - `mixed`: Some combination of the other methods, usually in some iterative process.
    - `team`: One of more individuals make decisions about term inclusions, categories, and weights.
  - `subject`: Broad, rough subject or purpose of the dictionary:
    - `emotion`: Terms relate to emotions, potentially exemplifying or expressing them.
    - `general`: A large range of categories, aiming to capture the content of the text.

    - impression: Terms are categorized and weighted based on the impression they might give.
    - language: Terms are categorized or weighted based on their linguistic features, such as part of speech, specificity, or area of use.
    - social: Terms relate to social phenomena, such as characteristics or concerns of social entities.
  - terms: Number of unique terms across categories.
  - term_type: Format of the terms:
    - glob: Include asterisks which denote inclusion of any characters until a word boundary.
    - glob+: Glob-style asterisks with regular expressions within terms.
    - ngram: Includes any number of words as a term, separated by spaces.
    - pattern: A string of characters, potentially within or between words, or spanning words.
    - regex: Regular expressions.
    - stem: Unigrams with common endings removed.
    - unigram: Complete single words.
  - weighted: Indicates whether weights are associated with terms. This determines the file type of the dictionary: dictionaries with weights are stored as .csv, and those without are stored as .dic files.
  - regex_characters: Logical indicating whether special regular expression characters are present in any term, which might need to be escaped if the terms are used in regular expressions. Glob-type terms allow complete parens (at least one open and one closed, indicating preceding or following words), and initial and terminal asterisks. For all other terms, `[](){}*.^$+?\|` are counted as regex characters. These could be escaped in R with `gsub('([][](){*.^$+?\\|])','\\\\1',terms)` if `terms` is a character vector, and in Python with (importing re) `[re.sub(r'([][(){*.^$+?\|])',r'\\1',term) for term in terms]` if `terms` is a list.
  - categories: Category names in the order in which they appear in the dictionary file, separated by commas.
  - ncategories: Number of categories.
  - original_max: Maximum value of the original dictionary before standardization: `original values / max(original values) * 100`. Dictionaries with no weights are considered to have a max of 1.
  - osf: ID of the file on OSF, translating to the file's URL: https://osf.io/osf.
  - wiki: URL of the dictionary's wiki.
  - downloaded: Path to the file if downloaded, and '' otherwise.
- selected: A subset of info selected by query.

See Also

Other Dictionary functions: download.dict(), lma_patcat(), lma_termcat(), read.dic()

Examples

```
# just retrieve information about available dictionaries
```

```
(dicts = select.dict()$info)

# select all dictionaries mentioning sentiment or emotion
(sentiment_dicts = select.dict('sentiment emotion')$selected)
```

`select.lspace` *Select Latent Semantic Spaces*

Description

Retrieve information and links to latent semantic spaces (sets of word vectors/embeddings) available at osf.io/489he, and optionally download their term mappings (osf.io/xr7jv).

Usage

```
select.lspace(query = NULL, dir = getOption("lingmatch.lspace.dir"),
  get.map = FALSE, check.md5 = TRUE, mode = "wb")
```

Arguments

`query`	A character matching a space name, or a character vector of terms, used to select spaces. If length is over 1, `get.map` is set to `TRUE`.
`dir`	Path to a directory containing `lma_term_map.rda` and downloaded spaces; will look in `getOption('lingmatch.lspace.dir')` and `'~/Latent Semantic Spaces'` by default.
`get.map`	Logical; if `TRUE` and `lma_term_map.rda` is not found in `dir`, the term map (lma_term_map.rda) is downloaded and decompressed.
`check.md5`	Logical; if `TRUE` (default), retrieves the MD5 checksum from OSF, and compares it with that calculated from the downloaded file to check its integrity.
`mode`	Passed to `download.file` when downloading the term map.

Value

A list with varying entries:

- `info`: The version of osf.io/9yzca stored internally; a `data.frame` with spaces as row names, and information about each space in columns:
  - `terms`: number of terms in the space
  - `corpus`: corpus(es) on which the space was trained
  - `model`: model from which the space was trained
  - `dimensions`: number of dimensions in the model (columns of the space)
  - `model_info`: some parameter details about the model
  - `original_max`: maximum value used to normalize the space; the original space would be `(vectors * original_max) / 100`
  - `osf_dat`: OSF id for the `.dat` files; the URL would be https://osf.io/`osf_dat`

  - osf_terms: OSF id for the _terms.txt files; the URL would be https://osf.io/osf_terms
  - wiki: link to the wiki for the space
  - downloaded: path to the .dat file if downloaded, and '' otherwise.
- selected: A subset of info selected by query.
- term_map: If get.map is TRUE or lma_term_map.rda is found in dir, a copy of osf.io/xr7jv, which has space names as column names, terms as row names, and indices as values, with 0 indicating the term is not present in the associated space.

See Also

Other Latent Semantic Space functions: download.lspace(), lma_lspace(), standardize.lspace()

Examples

```
# just retrieve information about available spaces
(spaces = select.lspace())

# retrieve all spaces that used word2vec
(w2v_spaces = select.lspace('word2vec')$selected)

## Not run:

# select spaces by terms
select.lspace(c(
  'part-time', 'i/o', "'cause", 'brexit', 'debuffs'
))$selected[, c('terms', 'coverage')]


## End(Not run)
```

---

standardize.lspace *Standardize a Latent Semantic Space*

Description

Reformat a .rda file which has a matrix with terms as row names, or a plain-text embeddings file which has a term at the start of each line, and consistent delimiting characters. Plain-text files are processed line-by-line, so large spaces can be reformatted RAM-conservatively.

Usage

```
standardize.lspace(infile, name, sep = " ", digits = 9,
  dir = getOption("lingmatch.lspace.dir"), outdir = dir, remove = "",
  term_check = "^[a-zA-Z]+$|^['a-zA-Z][a-zA-Z.'\\/-]*[a-zA-Z.]$",
  verbose = FALSE)
```

Arguments

infile	Name of the .rda or plain-text file relative to dir, e.g., "default.rda" or "glove/glove.6B.300d.txt".
name	Base name of the reformatted file and term file; e.g., "glove" would result in glove.dat and glove_terms.txt in outdir.
sep	Delimiting character between values in each line, e.g., " " or "\t". Only applies to plain-text files.
digits	Number of digits to round values to; default is 9.
dir	Path to folder containing infiles. Default is getOption('lingmatch.lspace.dir'), which must be set in the current session. If this is not specified and infile is a full path, dir will be set to infile's parent directory.
outdir	Path to folder in which to save standardized files; default is dir.
remove	A string with a regex pattern to be removed from term names (i.e., gsub(remove, "",term)); default is "", which is ignored.
term_check	A string with a regex pattern by which to filter terms; i.e., only lines with fully matched terms are written to the reformatted file. The default attempts to retain only regular words, including those with dashes, foreword slashes, and periods. Set to an empty string ("") to write all lines regardless of term.
verbose	Logical: if TRUE, prints the current line number and its term to the console every 1,000 lines. Only applies to plain-text files.

Value

Path to the standardized [1] data file and [2] terms file if applicable.

See Also

Other Latent Semantic Space functions: download.lspace(), lma_lspace(), select.lspace()

Examples

```
## Not run:

# from https://sites.google.com/site/fritzgntr/software-resources/semantic_spaces
standardize.lspace('EN_100k_lsa.rda', '100k_lsa')

# from https://fasttext.cc/docs/en/english-vectors.html
standardize.lspace('crawl-300d-2M.vec', 'facebook_crawl')

# Standardized versions of these spaces can also be downloaded with download.lspace.


## End(Not run)
```

Index