Package ‘text2map’

April 11, 2024

Type Package

Title R Tools for Text Matrices, Embeddings, and Networks

Version 0.2.0

Description This is a collection of functions optimized for working with
with various kinds of text matrices. Focusing on
the text matrix as the primary object - represented
either as a base R dense matrix or a 'Matrix' package sparse
matrix - allows for a consistent and intuitive interface
that stays close to the underlying mathematical foundation
of computational text analysis. In particular, the package
includes functions for working with word embeddings,
text networks, and document-term matrices. Methods developed in
Stoltz and Taylor (2019) <doi:10.1007/s42001-019-00048-6>,
Taylor and Stoltz (2020) <doi:10.1007/s42001-020-00075-8>,
Taylor and Stoltz (2020) <doi:10.15195/v7.a23>, and
Stoltz and Taylor (2021) <doi:10.1016/j.poetic.2021.101567>.

URL `https://gitlab.com/culturalcartography/text2map`

License MIT + file LICENSE

Encoding UTF-8

LazyData true

BugReports `https://gitlab.com/culturalcartography/text2map/-/issues`

RoxygenNote 7.3.1

Depends R (>= 3.5.0), Matrix (>= 1.4.2)

Imports text2vec, parallel, doParallel, foreach, stringi, dplyr, kit,
fastmatch, methods, qgraph (>= 1.6.9), igraph (>= 1.2.6),
rlang, ClusterR, tibble, rsvd, permute

Suggests testthat (>= 3.0.0), tm, quanteda

Config/testthat/edition 3

Config/Needs/website rmarkdown

NeedsCompilation no

Author Dustin Stoltz [aut, cre] (<https://orcid.org/0000-0002-4774-0765>),
Marshall Taylor [aut] (<https://orcid.org/0000-0002-7440-0723>)

Maintainer Dustin Stoltz <dss219@lehigh.edu>

Repository CRAN

Date/Publication 2024-04-11 08:10:02 UTC

R topics documented:

`anchor_lists` *A dataset of anchor lists*

Description

A dataset containing juxtaposing pairs of English words for 26 semantic relations. These anchors are used with the `get_anchors()` function, which can then be used with the `get_direction()` function. These have been collected from previously published articles and should be used as a starting point for defining a given relation in a word embedding model.

Usage

```
anchor_lists
```

Format

A data frame with 303 rows and 4 variables.

Variables

Variables:

- add. words to be added (or the positive direction)
- subtract. words to be subtract (or the negative direction)
- relation. the relation to be extracted, 26 relations available
- domain. 6 broader categories within which each relation falls

See Also

CoCA, get_direction, get_centroid, get_anchors

`CMDist` *Calculate Concept Mover's Distance*

Description

Concept Mover's Distance classifies documents of any length along a continuous measure of engagement with a given concept of interest using word embeddings.

Usage

```
CMDist(
  dtm,
  cw = NULL,
  cv = NULL,
  wv,
  missing = "stop",
  scale = TRUE,
  sens_interval = FALSE,
  alpha = 1,
  n_iters = 20L,
  parallel = FALSE,
  threads = 2L,
  setup_timeout = 120L
)

cmdist(
  dtm,
  cw = NULL,
  cv = NULL,
  wv,
  missing = "stop",
  scale = TRUE,
  sens_interval = FALSE,
  alpha = 1,
  n_iters = 20L,
  parallel = FALSE,
  threads = 2L,
  setup_timeout = 120L
)
```

Arguments

`dtm`	Document-term matrix with words as columns. Works with DTMs produced by any popular text analysis package, or using the `dtm_builder()` function.
`cw`	Vector with concept word(s) (e.g., `c("love", "money")`, `c("critical thinking")`)
`cv`	Concept vector(s) as output from `get_direction()`, `get_centroid()`, or `get_regions()`
`wv`	Matrix of word embedding vectors (a.k.a embedding model) with rows as words.
`missing`	Indicates what action to take if words are not in embeddings. If `action = "stop"` (default), the function is stopped and an error messages states which words are missing. If `action = "remove"`, output is the same as terms but missing words or rows with missing words are removed. Missing words will be printed as a message.
`scale`	Logical (default = `FALSE`) uses `scale()` on output. This will set zero to the mean of the estimates, and scale by the standard deviation of the estimates. Document estimates will, therefore, be relative to other documents within that specific run, but not necessarily across discrete runs.

`sens_interval`	logical (default = `FALSE`), if `TRUE` several CMDs will be estimate on N resampled DTMs, sensitivity intervals are produced by returning the 2.5 and 97.5 percentiles of estimated CMDs for a given concept word or concept vector.
`alpha`	If `sens_interval` = `TRUE`, a number indicating the proportion of the document length to be resampled for sensitivity intervals. Default is 1 or 100 percent of each documents' length.
`n_iters`	If `sens_interval` = `TRUE`, integer (default = 20L) indicates the number of resampled DTMs to produced for sensitivity intervals
`parallel`	Logical (default = `FALSE`), whether to parallelize estimate
`threads`	If `parallel` = `TRUE`, an integer indicating attempts to connect to master before failing.
`setup_timeout`	If `parallel` = `TRUE`, maximum number of seconds a worker attempts to connect to master before failing.

Details

`CMDist()` requires three things: a (1) document-term matrix (DTM), a (2) matrix of word embedding vectors, and (3) concept words or concept vectors. The function uses *word counts* from the DTM and *word similarities* from the cosine similarity of their respective word vectors in a word embedding model. The "cost" of transporting all the words in a document to a single vector or a few vectors (denoting a concept of interest) is the measure of engagement, with higher costs indicating less engagement. For intuitiveness the output of `CMDist()` is inverted such that higher numbers will indicate *more engagement* with a concept of interest.

The vector, or vectors, of the concept are specified in several ways. The simplest involves selecting a single word from the word embeddings, the analyst can also specify the concept by indicating a few words. The algorithm then splits the overall flow between each concept word (roughly) depending on which word in the document is nearest. The words need not be in the DTM, but they must be in the word embeddings (the function will either stop or remove words not in the embeddings).

Instead of selecting a word already in the embedding space, the function can also take a vector extracted from the embedding space in the form of a centroid (which averages the vectors of several words) ,a direction (which uses the offset of several juxtaposing words), or a region (which is built by clustering words into $k$ regions). The `get_centroid()`, `get_direction()`, and `get_regions()` functions will extract these.

Value

Returns a data frame with the first column as document ids and each subsequent column as the CMD engagement corresponding to each concept word or concept vector. The upper and lower bound estimates will follow each unique CMD if `sens_interval` = `TRUE`.

Author(s)

Dustin Stoltz and Marshall Taylor

References

Stoltz, Dustin S., and Marshall A. Taylor. (2019) 'Concept Mover's Distance' *Journal of Computational Social Science* 2(2):293-313. doi:10.1007/s42001019000486.
Taylor, Marshall A., and Dustin S. Stoltz. (2020) 'Integrating semantic directions with concept mover's distance to measure binary concept engagement.' *Journal of Computational Social Science* 1-12. doi:10.1007/s42001020000758.
Taylor, Marshall A., and Dustin S. Stoltz. (2020) 'Concept Class Analysis: A Method for Identifying Cultural Schemas in Texts.' *Sociological Science* 7:544-569. doi:10.15195/v7.a23.

See Also

CoCA, get_direction, get_centroid

Examples

```
# load example word embeddings
data(ft_wv_sample)

# load example text
data(jfk_speech)

# minimal preprocessing
jfk_speech$sentence <- tolower(jfk_speech$sentence)
jfk_speech$sentence <- gsub("[[:punct:]]+", " ", jfk_speech$sentence)

# create DTM
dtm <- dtm_builder(jfk_speech, sentence, sentence_id)

# example 1
cm.dists <- CMDist(dtm,
  cw = "space",
  wv = ft_wv_sample
)

# example 2
space <- c("spacecraft", "rocket", "moon")
cen <- get_centroid(anchors = space, wv = ft_wv_sample)

cm.dists <- CMDist(dtm,
  cv = cen,
  wv = ft_wv_sample
)
```

CoCA *Performs Concept Class Analysis (CoCA)*

Description

CoCA outputs schematic classes derived from documents' engagement with multiple bi-polar concepts (in a Likert-style fashion). The function requires a (1) DTM of a corpus which can be obtained using any popular text analysis package, or from the `dtm_builder()` function, and (2) semantic directions as output from the `get_direction()`. `CMDist()` works under the hood. Code modified from the `corclass` package.

Usage

```
CoCA(
  dtm,
  wv = NULL,
  directions = NULL,
  filter_sig = TRUE,
  filter_value = 0.05,
  zero_action = c("drop", "ownclass")
)

coca(
  dtm,
  wv = NULL,
  directions = NULL,
  filter_sig = TRUE,
  filter_value = 0.05,
  zero_action = c("drop", "ownclass")
)
```

Arguments

`dtm`	Document-term matrix with words as columns. Works with DTMs produced by any popular text analysis package, or you can use the `dtm_builder()` function.
`wv`	Matrix of word embedding vectors (a.k.a embedding model) with rows as words.
`directions`	direction vectors output from get_direction()
`filter_sig`	logical (default = TRUE), sets 'insignificant' ties to 0 to decrease noise and increase stability
`filter_value`	Minimum significance cutoff. Absolute row correlations below this value will be set to 0
`zero_action`	If 'drop', CCA drops rows with 0 variance from the analyses (default). If 'ownclass', the correlations between 0-variance rows and all other rows is set 0, and the correlations between all pairs of 0-var rows are set to 1

Value

Returns a named list object of class `CoCA`. List elements include:

- membership: document memberships
- modules: schematic classes
- cormat: correlation matrix

Author(s)

Dustin Stoltz and Marshall Taylor

References

Taylor, Marshall A., and Dustin S. Stoltz. (2020) 'Concept Class Analysis: A Method for Identifying Cultural Schemas in Texts.' *Sociological Science* 7:544-569. doi:10.15195/v7.a23.
Boutyline, Andrei. 'Improving the measurement of shared cultural schemas with correlational class analysis: Theory and method.' Sociological Science 4.15 (2017): 353-393. doi:10.15195/v4.a15

See Also

CMDist, get_direction

Examples

```
#' # load example word embeddings
data(ft_wv_sample)

# load example text
data(jfk_speech)

# minimal preprocessing
jfk_speech$sentence <- tolower(jfk_speech$sentence)
jfk_speech$sentence <- gsub("[[:punct:]]+", " ", jfk_speech$sentence)

# create DTM
dtm <- dtm_builder(jfk_speech, sentence, sentence_id)

# create semantic directions
gen <- data.frame(
  add = c("woman"),
  subtract = c("man")
)

die <- data.frame(
  add = c("alive"),
  subtract = c("die")
)

gen_dir <- get_direction(anchors = gen, wv = ft_wv_sample)
die_dir <- get_direction(anchors = die, wv = ft_wv_sample)

sem_dirs <- rbind(gen_dir, die_dir)

classes <- CoCA(
  dtm = dtm,
  wv = ft_wv_sample,
  directions = sem_dirs,
  filter_sig = TRUE,
  filter_value = 0.05,
  zero_action = "drop"
)

print(classes)
```

`doc_centrality` *Find a specified document centrality metric*

Description

Given a document-term matrix or a document-similarity matrix, this function returns specified text network-based centrality measures. Currently, this includes degree, eigenvector, betweenness, and spanning.

Usage

```
doc_centrality(mat, method, alpha = 1L, two_mode = TRUE)
```

Arguments

`mat`	Document-term matrix with terms as columns or a document-similarity matrix with documents as rows and columns.
`method`	Character vector indicating centrality method, including "degree", "eigen", "span", and "between".
`alpha`	Number (default = 1) indicating the tuning parameter for weighted metrics.
`two_mode`	Logical (default = TRUE), indicating whether the input matrix is two mode (i.e. a document-term matrix) or one-mode (i.e. document-similarity matrix)

Details

If a document-term matrix is provided, the function obtains the one-mode document-level projection to get the document-similarity matrix using `tcrossprod()`. If a one-mode document-similarity matrix is provided, then this step is skipped. This way document similiarities may be obtained using other methods, such as Word-Mover's Distance (see `doc_similarity`). The diagonal is ignored in all calculations.

Document centrality methods include:

- degree: Opsahl's weighted degree centrality with tuning parameter "alpha"
- between: vertex betweenness centrality using Brandes' method
- eigen: eigenvector centrality using Freeman's method
- span: Modified Burt's constraint following Stoltz and Taylor's method, uses a tuning parameter "alpha" and the output is scaled.

Value

A dataframe with two columns

Author(s)

Dustin Stoltz

References

Brandes, Ulrik (2000) 'A faster algorithm for betweenness centrality' *Journal of Mathematical Sociology*. 25(2):163-177 doi:10.1080/0022250X.2001.9990249.
Opsahl, Tore, et al. (2010) 'Node centrality in weighted networks: Generalizing degree and shortest paths.' *Social Networks*. 32(3)245:251 doi:10.1016/j.socnet.2010.03.006
Stoltz, Dustin; Taylor, Marshall (2019) 'Textual Spanning: Finding Discursive Holes in Text Networks' *Socius*. doi:10.1177/2378023119827674

Examples

```
# load example text
data(jfk_speech)

# minimal preprocessing
jfk_speech$sentence <- tolower(jfk_speech$sentence)
jfk_speech$sentence <- gsub("[[:punct:]]+", " ", jfk_speech$sentence)

# create DTM
dtm <- dtm_builder(jfk_speech, sentence, sentence_id)

ddeg <- doc_centrality(dtm, method = "degree")
deig <- doc_centrality(dtm, method = "eigen")
dbet <- doc_centrality(dtm, method = "between")
dspa <- doc_centrality(dtm, method = "span")

# with a document-similarity matrix (dsm)

dsm <- doc_similarity(dtm, method = "cosine")
ddeg <- doc_centrality(dsm, method = "degree", two_mode = FALSE)
```

doc_similarity *Find a similarities between documents*

Description

Given a document-term matrix (DTM) this function returns the similarities between documents using a specified method (see details). The result is a square document-by-document similarity matrix (DSM), equivalent to a weighted adjacency matrix in network analysis.

Usage

```
doc_similarity(x, y = NULL, method, wv = NULL)
```

Arguments

`x`	Document-term matrix with terms as columns.
`y`	Optional second matrix (default = `NULL`).
`method`	Character vector indicating similarity method, including projection, cosine, wmd, and centroid (see Details).
`wv`	Matrix of word embedding vectors (a.k.a embedding model) with rows as words. Required for "wmd" and "centroid" similarities.

Details

Document similarity methods include:

- projection: finds the one-mode projection matrix from the two-mode DTM using `tcrossprod()` which measures the shared vocabulary overlap
- cosine: compares row vectors using cosine similarity
- jaccard: compares proportion of common words to unique words in both documents
- wmd: word mover's distance to compare documents (requires word embedding vectors), using linear-complexity relaxed word mover's distance
- centroid: represents each document as a centroid of their respective vocabulary, then uses cosine similarity to compare centroid vectors (requires word embedding vectors)

Author(s)

Dustin Stoltz

Examples

```
# load example word embeddings
data(ft_wv_sample)

# load example text
data(jfk_speech)

# minimal preprocessing
jfk_speech$sentence <- tolower(jfk_speech$sentence)
jfk_speech$sentence <- gsub("[[:punct:]]+", " ", jfk_speech$sentence)

# create DTM
dtm <- dtm_builder(jfk_speech, sentence, sentence_id)

dsm_prj <- doc_similarity(dtm, method = "projection")
dsm_cos <- doc_similarity(dtm, method = "cosine")
dsm_wmd <- doc_similarity(dtm, method = "wmd", wv = ft_wv_sample)
dsm_cen <- doc_similarity(dtm, method = "centroid", wv = ft_wv_sample)
```

dtm_builder *A fast unigram DTM builder*

Description

A streamlined function to take raw texts from a column of a data.frame and produce a sparse Document-Term Matrix (of generic class "dgCMatrix").

Usage

```
dtm_builder(
  data,
  text,
  doc_id = NULL,
  vocab = NULL,
  chunk = NULL,
  dense = FALSE,
  omit_empty = FALSE
)
```

Arguments

`data`	Data.frame with column of texts and column of document ids
`text`	Name of the column with documents' text
`doc_id`	Name of the column with documents' unique ids.
`vocab`	Default is `NULL`, if a list of terms is provided, the function will return a DTM with terms restricted to this vocabulary. Columns will also be in the same order as the list of terms.
`chunk`	Default is `NULL`, if an integer is provided, the function will "re-chunk" the corpus into new documents of a particular length. For example, `100L` will divide the corpus into new documents with 100 terms (with the final document likely including slightly less than 100).
`dense`	The default (`FALSE`) is to return a matrix of class "dgCMatrix" as DTMs typically have mostly zero cells. This is much more memory efficient. Setting dense to `TRUE` will return a normal base `R` matrix.
`omit_empty`	Logical (default = `FALSE`) indicating whether to omit rows that are empty after stopping any terms.

Details

The function is fast because it has few bells and whistles:

- No weighting schemes other than raw counts
- Tokenizes by the fixed, single whitespace
- Only tokenizes unigrams. No bigrams, trigrams, etc...

- Columns are in the order unique terms are discovered
- No preprocessing during building
- Outputs a basic sparse Matrix or dense matrix

Weighting or stopping terms can be done efficiently after the fact with simple matrix operations, rather than achieved implicitly within the function itself. For example, using the `dtm_stopper()` function. Prior to creating the DTM, texts should have whitespace trimmed, if desired, punctuation removed and terms lowercased.

Like `tidytext`'s DTM functions, `dtm_builder()` is optimized for use in a pipeline, but unlike `tidytext`, it does not build an intermediary tripletlist, so `dtm_builder()` is faster and far more memory efficient.

The function can also `chunk` the corpus into documents of a given length (default is `NULL`). If the integer provided is `200L`, this will divide the corpus into new documents with 200 terms (with the final document likely including slightly less than 200). If the total terms in the corpus were less than or equal to `chunk` integer, this would produce a DTM with one document (most will probably not want this).

If the vocabulary is already known, or standardizing vocabulary across several DTMs is desired, a list of terms can be provided to the `vocab` argument. Columns of the DTM will be in the order of the list of terms.

Value

returns a document-term matrix of class "dgCMatrix" or class "matrix"

Author(s)

Dustin Stoltz

Examples

```
library(dplyr)

my_corpus <- data.frame(
  text = c(
    "I hear babies crying I watch them grow",
    "They'll learn much more than I'll ever know",
    "And I think to myself",
    "What a wonderful world",
    "Yes I think to myself",
    "What a wonderful world"
  ),
  line_id = paste0("line", seq_len(6))
)
## some text preprocessing
my_corpus$clean_text <- tolower(gsub("'", "", my_corpus$text))

# example 1 with R 4.1 pipe

dtm <- my_corpus |>
```

```
  dtm_builder(clean_text, line_id)


# example 2 without pipe
dtm <- dtm_builder(
  data = my_corpus,
  text = clean_text,
  doc_id = line_id
)

# example 3 with dplyr pipe and mutate

dtm <- my_corpus %>%
  mutate(
    clean_text = gsub("'", "", text),
    clean_text = tolower(clean_text)
  ) %>%
  dtm_builder(clean_text, line_id)

# example 4 with dplyr and chunk of 3 terms
dtm <- my_corpus %>%
  dtm_builder(clean_text,
    line_id,
    chunk = 3L
  )


# example 5 with user defined vocabulary
my.vocab <- c("wonderful", "world", "haiku", "think")

dtm <- dtm_builder(
  data = my_corpus,
  text = clean_text,
  doc_id = line_id,
  vocab = my.vocab
)
```

`dtm_melter` *Melt a DTM into a triplet data frame*

Description

Converts a DTM into a data frame with three columns: documents, terms, frequency. Each row is a unique document by term frequency. This is akin to `reshape2` packages `melt` function, but works on a sparse matrix. The resulting data frame is also equivalent to the `tidytext` triplet tibble.

Usage

```
dtm_melter(dtm)
```

Arguments

`dtm`	Document-term matrix with terms as columns. Works with DTMs produced by any popular text analysis package, or using the `dtm_builder()` function.

Value

returns data frame with three columns: doc_id, term, freq

Author(s)

Dustin Stoltz

`dtm_resampler` *Resamples an input DTM to generate new DTMs*

Description

Takes any DTM and randomly resamples from each row, creating a new DTM

Usage

```
dtm_resampler(dtm, alpha = NULL, n = NULL)
```

Arguments

`dtm`	Document-term matrix with terms as columns. Works with DTMs produced by any popular text analysis package, or you can use the `dtm_builder()` function.
`alpha`	Number indicating proportion of document lengths, e.g., `alpha = 1` returns resampled rows that are the same lengths as the original DTM.
`n`	Integer indicating the length of documents to be returned, e.g., `n = 100L` will bring documents shorter than 100 tokens up to 100, while bringing documents longer than 100 tokens down to 100.

Details

Using the row counts as probabilities, each document's tokens are resampled with replacement up to a certain proportion of the row count (set by alpha). This function can be used with iteration to "bootstrap" a DTM without returning to the raw text. It does not iterate, however, so operations can be performed on one DTM at a time without storing multiple DTMs in memory.

If `alpha` is less than 1, then a proportion of each documents' lengths is returned. For example, `alpha = 0.50` will return a resampled DTM where each row has half the tokens of the original DTM. If `alpha = 2`, than each row in the resampled DTM twice the number of tokens of the original DTM. If an integer is provided to `n` then all documents will be resampled to that length. For example, `n = 2000L` will resample each document until they are 2000 tokens long – meaning those shorter than 2000 will be increased in length, while those longer than 2000 will be decreased in length. `alpha` and `n` should not be specified at the same time.

Value

returns a document-term matrix of class "dgCMatrix"

`dtm_stats` *Gets DTM summary statistics*

Description

`dtm_stats()` provides a summary of corpus-level statistics using any document-term matrix. These include (1) basic information on size (total documents, total unique terms, total tokens), (2) lexical richness, (3) distribution information, (4) central tendency, and (5) character-level information.

Usage

```
dtm_stats(
  dtm,
  richness = TRUE,
  distribution = TRUE,
  central = TRUE,
  character = TRUE,
  simplify = FALSE
)
```

Arguments

`dtm`	Document-term matrix with terms as columns. Works with DTMs produced by any popular text analysis package, or you can use the `dtm_builder()` function.
`richness`	Logical (default = TRUE), whether to include statistics about lexical richness, i.e. terms that occur once, twice, and three times (hapax, dis, tris), and the total type-token ratio.
`distribution`	Logical (default = TRUE), whether to include statistics about the distribution, i.e. min, max st. dev, skewness, kurtosis.
`central`	Logical (default = TRUE), whether to include statistics about the central tendencies i.e. mean and median for types and tokens.
`character`	Logical (default = TRUE), whether to include statistics about the character lengths of terms, i.e. min, max, mean
`simplify`	Logical (default = FALSE), whether to return statistics as a data frame where each statistic is a column. Default returns a list of small data frames.

Value

A list of one to five data frames with summary statistics (if `simplify=FALSE`), otherwise a single data frame where each statistic is a column.

Author(s)

Dustin Stoltz

`dtm_stopper` *Removes terms from a DTM based on rules*

Description

`dtm_stopper` will "stop" terms from the analysis by removing columns in a DTM based on stop rules. Rules include matching terms in a precompiled or custom list, terms meeting an upper or lower document frequency threshold, or terms meeting an upper or lower term frequency threshold.

Usage

```
dtm_stopper(
  dtm,
  stop_list = NULL,
  stop_termfreq = NULL,
  stop_termrank = NULL,
  stop_termprop = NULL,
  stop_docfreq = NULL,
  stop_docprop = NULL,
  stop_hapax = FALSE,
  stop_null = FALSE,
  omit_empty = FALSE,
  dense = FALSE,
  ignore_case = TRUE
)
```

Arguments

`dtm`	Document-term matrix with terms as columns. Works with DTMs produced by any popular text analysis package, or you can use the `dtm_builder` function.
`stop_list`	Vector of terms, from a precompiled stoplist or custom list such as `c("never", "gonna", "give")`.
`stop_termfreq`	Vector of two numbers indicating the lower and upper threshold for exclusion (see details). Use `Inf` for max or min, respectively.
`stop_termrank`	Single integer indicating upper term rank threshold for exclusion (see details).
`stop_termprop`	Vector of two numbers indicating the lower and upper threshold for exclusion (see details). Use `Inf` for max or min, respectively.
`stop_docfreq`	Vector of two numbers indicating the lower and upper threshold for exclusion (see details). Use `Inf` for max or min, respectively.
`stop_docprop`	Vector of two numbers indicating the lower and upper threshold for exclusion (see details). Use `Inf` for max or min, respectively.
`stop_hapax`	Logical (default = FALSE) indicating whether to remove terms occurring one time (or zero times), a.k.a. hapax legomena
`stop_null`	Logical (default = FALSE) indicating whether to remove terms that occur zero times in the DTM.

`omit_empty`	Logical (default = FALSE) indicating whether to omit rows that are empty after stopping any terms.
`dense`	The default (`FALSE`) is to return a matrix of class "dgCMatrix". Setting dense to `TRUE` will return a normal base `R` dense matrix.
`ignore_case`	Logical (default = TRUE) indicating whether to ignore capitalization.

Details

Stopping terms by removing their respective columns in the DTM is significantly more efficient than searching raw text with string matching and deletion rules. Behind the scenes, the function relies on the `fastmatch` package to quickly match/not-match terms.

The `stop_list` arguments takes a list of terms which are matched and removed from the DTM. If `ignore_case = TRUE` (the default) then word case will be ignored.

The `stop_termfreq` argument provides rules based on a term's occurrences in the DTM as a whole – regardless of its within document frequency. If real numbers between 0 and 1 are provided then terms will be removed by corpus proportion. For example `c(0.01, 0.99)`, terms that are either below 1% of the total tokens or above 99% of the total tokens will be removed. If integers are provided then terms will be removed by total count. For example `c(100, 9000)`, occurring less than 100 or more than 9000 times in the corpus will be removed. This also means that if `c(0, 1)` is provided, then the will only *keep* terms occurring once.

The `stop_termrank` argument provides the upper threshold for a terms' rank in the corpus. For example, `5L` will remove the five most frequent terms.

The `stop_docfreq` argument provides rules based on a term's document frequency – i.e. the number of documents within which it occurs, regardless of how many times it occurs. If real numbers between 0 and 1 are provided then terms will be removed by corpus proportion. For example `c(0.01, 0.99)`, terms in more than 99% of all documents or terms that are in less than 1% of all documents. For example `c(100, 9000)`, then words occurring in less than 100 documents or more than 9000 documents will be removed. This means that if `c(0, 1)` is provided, then the function will only *keep* terms occurring in exactly one document, and remove terms in more than one.

The `stop_hapax` argument is a shortcut for removing terms occurring just one time in the corpus – called hapax legomena. Typically, a size-able portion of the corpus tends to be hapax terms, and removing them is a quick solution to reducing the dimensions of a DTM. The DTM must be frequency counts (not relative frequencies).

The `stop_null` argument removes terms that do not occur at all. In other words, there is a column for the term, but the entire column is zero. This can occur for a variety of reasons, such as starting with a predefined vocabulary (e.g., using dtm_builder's `vocab` argument) or through some cleaning processes.

The `omit_empty` argument will remove documents that are empty

Value

returns a document-term matrix of class "dgCMatrix"

Author(s)

Dustin Stoltz

Examples

```
# create corpus and DTM
my_corpus <- data.frame(
  text = c(
    "I hear babies crying I watch them grow",
    "They'll learn much more than I'll ever know",
    "And I think to myself",
    "What a wonderful world",
    "Yes I think to myself",
    "What a wonderful world"
  ),
  line_id = paste0("line", seq_len(6))
)
## some text preprocessing
my_corpus$clean_text <- tolower(gsub("'", "", my_corpus$text))

dtm <- dtm_builder(
  data = my_corpus,
  text = clean_text,
  doc_id = line_id
)

## example 1 with R 4.1 pipe

dtm_st <- dtm |>
  dtm_stopper(stop_list = c("world", "babies"))


## example 2 without pipe
dtm_st <- dtm_stopper(
  dtm,
  stop_list = c("world", "babies")
)

## example 3 precompiled stoplist
dtm_st <- dtm_stopper(
  dtm,
  stop_list = get_stoplist("snowball2014")
)

## example 4, stop top 2
dtm_st <- dtm_stopper(
  dtm,
  stop_termrank = 2L
)

## example 5, stop docfreq
dtm_st <- dtm_stopper(
  dtm,
  stop_docfreq = c(2, 5)
)
```

`find_projection` *Find the 'projection matrix' to a semantic vector*

Description

"Project" each word in a word embedding matrix of D dimension along a vector of D dimensions, extracted from the same embedding space. The vector can be a single word, or a concept vector obtained from `get_centroid()`, `get_direction()`, or `get_regions()`.

Usage

```
find_projection(wv, vec)
```

Arguments

`wv`	Matrix of word embedding vectors (a.k.a embedding model) with rows as words.
`vec`	Vector extracted from the embeddings

Details

All the vectors in the matrix A are projected onto the a vector, v, to find the projection matrix, P, defined as:

$$P = \frac{A \cdot v}{v \cdot v} * v$$

Value

A new word embedding matrix, each row of which is parallel to vector.

`find_rejection` *Find the 'rejection matrix' from a semantic vector*

Description

"Reject" each word in a word embedding matrix of D dimension from a vector of D dimensions, extracted from the same embedding space. The vector can be a single word, or a concept vector obtained from `get_centroid()`, `get_direction()`, or `get_regions()`.

Usage

```
find_rejection(wv, vec)
```

Arguments

`wv`	Matrix of word embedding vectors (a.k.a embedding model) with rows as words.
`vec`	Vector extracted from the embeddings

Value

A new word embedding matrix, each row of which is rejected from vector.

`find_transformation` *Find a specified matrix transformation*

Description

Given a matrix, B, of word embedding vectors (source) with terms as rows, this function finds a transformed matrix following a specified operation. These include: centering (i.e. translation) and normalization (i.e. scaling). In the first, B is centered by subtracting column means. In the second, B is normalized by the L2 norm. Both have been found to improve word embedding representations. The function also finds a transformed matrix that approximately aligns B, with another matrix, A, of word embedding vectors (reference), using Procrustes transformation (see details). Finally, given a term-co-occurrence matrix built on a local corpus, the function can "retrofit" pretrained embeddings to better match the local corpus.

Usage

```
find_transformation(
  wv,
  ref = NULL,
  method = c("align", "norm", "center", "retrofit")
)
```

Arguments

`wv`	Matrix of word embedding vectors (a.k.a embedding model) with rows as terms (the source matrix to be transformed).
`ref`	If `method = "align"`, this is the reference matrix toward which the source matrix is to be aligned.
`method`	Character vector indicating the method to use for the transformation. Current methods include: "align", "norm", "center", and "refrofit" – see details.

Details

Aligning a source matrix of word embedding vectors, B, to a reference matrix, A, has primarily been used as a post-processing step for embeddings trained on longitudinal corpora for diachronic analysis or for cross-lingual embeddings. Aligning preserves internal (cosine) distances, while orient the source embeddings to minimize the sum of squared distances (and is therefore a Least Squares problem). Alignment is accomplished with the following steps:

- translation: centering by column means
- scaling: scale (normalizes) by the L2 Norm
- rotation/reflection: rotates and a reflects to minimize sum of squared differences, using singular value decomposition

Alignment is asymmetrical, and only outputs the transformed source matrix, B. Therefore, it is typically recommended to align B to A, and then A to B. However, simplying centering and norming A after may be sufficient.

Value

A new word embedding matrix, transformed using the specified method.

References

Mikel Artetxe, Gorka Labaka, and Eneko Agirre. (2018). 'A robust self-learning method for fully unsupervised cross-lingual mappings of word embeddings.' *In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*. 789-798
Mikel Artetxe, Gorka Labaka, and Eneko Agirre. 2019. 'An effective approach to unsupervised machine translation.' *In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. 194-203
Hamilton, William L., Jure Leskovec, and Dan Jurafsky. (2018). 'Diachronic Word Embeddings Reveal Statistical Laws of Semantic Change.' `https://arxiv.org/abs/1605.09096v6`.
Lin, Zefeng, Xiaojun Wan, and Zongming Guo. (2019). 'Learning Diachronic Word Embeddings with Iterative Stable Information Alignment.' *Natural Language Processing and Chinese Computing*. 749-60. doi:10.1007/9783030322335_58.
Schlechtweg et al. (2019). 'A Wind of Change: Detecting and Evaluating Lexical Semantic Change across Times and Domains.' `https://arxiv.org/abs/1906.02979v1`. Shoemark et a. (2019). 'Room to Glo: A Systematic Comparison of Semantic Change Detection Approaches with Word Embeddings.' *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. 66-76. doi:10.18653/v1/D191007 Borg and Groenen. (1997). *Modern Multidimensional Scaling*. New York: Springer. 340-342

`ft_wv_sample` *Sample of fastText embeddings*

Description

These are a sample of the English fastText embeddings including 770 words matching those used in the `jfk_speech`. These are intended to be used for example code.

Usage

```
ft_wv_sample
```

Format

A matrix of 770 rows and 300 columns

`get_anchors` *Gets anchor terms from precompiled anchor lists*

Description

Produces a data.frame of juxtaposed word pairs used to extract a semantic direction from word embeddings. Can be used as input to `get_direction()`.

Usage

```
get_anchors(relation)
```

Arguments

`relation`	String indicating a semantic relation, 26 relations are available in the dataset (see details).

Details

Sets of juxtaposed "anchor" pairs are adapted from published work and associated with a particular semantic relation. These should be used as a starting point, not as a "ground truth."

Available relations include:

- activity
- affluence
- age
- attractiveness
- borders
- concreteness
- cultivation
- dominance
- education
- gender
- government
- purity
- safety
- sexuality
- skills
- status
- valence
- whiteness

Value

returns a tibble with two columns

Author(s)

Dustin Stoltz

Examples

```
gen <- get_anchors(relation = "gender")
```

`get_centroid` *Word embedding semantic centroid extractor*

Description

The function outputs an averaged vector from a set of anchor terms' word vectors. This average is roughly equivalent to the intersection of the contexts in which each word is used. This semantic centroid can be used for a variety of ends, and specifically as input to `CMDist()`. `get_centroid()` requires a list of terms, string of terms, data.frame or matrix. In the latter two cases, the first column will be used. The vectors are aggregated using the simple average. Terms can be repeated, and are therefore "weighted" by their counts.

Usage

```
get_centroid(anchors, wv, missing = "stop")
```

Arguments

`anchors`	List of terms to be averaged
`wv`	Matrix of word embedding vectors (a.k.a embedding model) with rows as words.
`missing`	what action to take if terms are not in embeddings. If action = "stop" (default), the function is stopped and an error messages states which terms are missing. If action = "remove", missing terms or rows with missing terms are removed. Missing terms will be printed as a message.

Value

returns a one row matrix

Author(s)

Dustin Stoltz

Examples

```
# load example word embeddings
data(ft_wv_sample)

space1 <- c("spacecraft", "rocket", "moon")

cen1 <- get_centroid(anchors = space1, wv = ft_wv_sample)

space2 <- c("spacecraft rocket moon")
cen2 <- get_centroid(anchors = space2, wv = ft_wv_sample)

identical(cen1, cen2)
```

`get_direction` *Word embedding semantic direction extractor*

Description

`get_direction()` outputs a vector corresponding to one pole of a "semantic direction" built from sets of antonyms or juxtaposed terms. The output can be used as an input to `CMDist()` and `CoCA()`. Anchors must be a two-column data.frame or a list of length == 2.

Usage

```
get_direction(anchors, wv, method = "paired", missing = "stop", n_dirs = 1L)
```

Arguments

`anchors`	A data frame or list of juxtaposed 'anchor' terms
`wv`	Matrix of word embedding vectors (a.k.a embedding model) with rows as terms.
`method`	Indicates the method used to generate vector offset. Default is 'paired'. See details.
`missing`	what action to take if terms are not in embeddings. If action = "stop" (default), the function is stopped and an error messages states which terms are missing. If action = "remove", missing terms or rows with missing terms are removed. Missing terms will be printed as a message.
`n_dirs`	If `method = "PCA"`, an integer indicating how many directions to return. Default = `1L`, indicating a single, bipolar, direction.

Details

Semantic directions can be estimated in using a few methods:

- 'paired' (default): each individual term is subtracted from exactly one other paired term. there must be the same number of terms for each side of the direction (although one word may be used more than once).

- ’pooled’: terms corresponding to one side of a direction are first averaged, and then these averaged vectors are subtracted. A different number of terms can be used for each side of the direction.
- ’L2’: the vector is calculated the same as with ’pooled’ but is then divided by the L2 ’Euclidean’ norm
- ’PCA’: vector offsets are calculated for each pair of terms, as with ’paired’, and if `n_dirs = 1L` (the default) then the direction is the first principal component. Users can return more than one direction by increasing the `n_dirs` parameter.

Value

returns a one row matrix

Author(s)

Dustin Stoltz

References

Bolukbasi, T., Chang, K. W., Zou, J., Saligrama, V., and Kalai, A. (2016). Quantifying and reducing stereotypes in word embeddings. arXiv preprint `https://arxiv.org/abs/1606.06121v1`.
Bolukbasi, Tolga, Kai-Wei Chang, James Zou, Venkatesh Saligrama, Adam Kalai (2016). ’Man Is to Computer Programmer as Woman Is to Homemaker? Debiasing Word Embeddings.’ Proceedings of the 30th International Conference on Neural Information Processing Systems. 4356-4364. `https://dl.acm.org/doi/10.5555/3157382.3157584`.
Taylor, Marshall A., and Dustin S. Stoltz. (2020) ’Concept Class Analysis: A Method for Identifying Cultural Schemas in Texts.’ *Sociological Science* 7:544-569. doi:10.15195/v7.a23.
Taylor, Marshall A., and Dustin S. Stoltz. (2020) ’Integrating semantic directions with concept mover’s distance to measure binary concept engagement.’ *Journal of Computational Social Science* 1-12. doi:10.1007/s42001020000758.
Kozlowski, Austin C., Matt Taddy, and James A. Evans. (2019). ’The geometry of culture: Analyzing the meanings of class through word embeddings.’ *American Sociological Review* 84(5):905-949. doi:10.1177/0003122419877135.
Arseniev-Koehler, Alina, and Jacob G. Foster. (2020). ’Machine learning as a model for cultural learning: Teaching an algorithm what it means to be fat.’ arXiv preprint `https://arxiv.org/abs/2003.12133v2`.

Examples

```
# load example word embeddings
data(ft_wv_sample)

# create anchor list
gen <- data.frame(
  add = c("woman"),
  subtract = c("man")
)
```

```
dir <- get_direction(anchors = gen, wv = ft_wv_sample)

dir <- get_direction(
  anchors = gen, wv = ft_wv_sample,
  method = "PCA", n = 1L
)
```

get_regions *Word embedding semantic region extractor*

Description

Given a set of word embeddings of d dimensions and v vocabulary, `get_regions()` finds k semantic regions in d dimensions. This, in effect, learns latent topics from an embedding space (a.k.a. topic modeling), which are directly comparable to both terms (with cosine similarity) and documents (with Concept Mover's distance using `CMDist()`).

Usage

```
get_regions(wv, k_regions = 5L, max_iter = 20L, seed = 0)
```

Arguments

`wv`	Matrix of word embedding vectors (a.k.a embedding model) with rows as words.
`k_regions`	Integer indicating the k number of regions to return
`max_iter`	Integer indicating the maximum number of iterations before k-means terminates.
`seed`	Integer indicating a random seed. Default is 0, which calls 'std::time(NULL)'.

Details

To group words into more encompassing "semantic regions" we use k-means clustering. We choose k-means primarily for it's ubiquity and the wide range of available diagnostic tools for k-means cluster.

A word embedding matrix of d dimensions and v vocabulary is "clustered" into k semantic regions which have d dimensions. Each region is represented by a single point defined by the d dimensional vector. The process discretely assigns all word vectors are assigned to a given region so as to minimize some error function, however as the resulting regions are in the same dimensions as the word embeddings, we can measure each terms similarity to each region. This, in effect, is a mixed membership topic model similar to topic modeling by Latent Dirichlet Allocation.

We use the `KMeans_arma` function from the `ClusterR` package which uses the Armadillo library.

Value

returns a matrix of class "dgCMatrix" with k rows and d dimensions

Author(s)

Dustin Stoltz

References

Butnaru, Andrei M., and Radu Tudor Ionescu. (2017) 'From image to text classification: A novel approach based on clustering word embeddings.' *Procedia computer science*. 112:1783-1792. doi:10.1016/j.procs.2017.08.211.
Zhang, Yi, Jie Lu, Feng Liu, Qian Liu, Alan Porter, Hongshu Chen, and Guangquan Zhang. (2018). 'Does Deep Learning Help Topic Extraction? A Kernel K-Means Clustering Method with Word Embedding.' *Journal of Informetrics*. 12(4):1099-1117. doi:10.1016/j.joi.2018.09.004.
Arseniev-Koehler, Alina and Cochran, Susan D and Mays, Vickie M and Chang, Kai-Wei and Foster, Jacob Gates (2021) 'Integrating topic modeling and word embedding to characterize violent deaths' doi:10.31235/osf.io/nkyaq

Examples

```
# load example word embeddings
data(ft_wv_sample)

my.regions <- get_regions(
  wv = ft_wv_sample,
  k_regions = 10L,
  max_iter = 10L,
  seed = 01984
)
```

`get_stoplist` *Gets stoplist from precompiled lists*

Description

Provides access to 8 precompiled stoplists, including the most commonly used stoplist from the Snowball stemming package ("snowball2014"), `text2map`'s tiny stoplist ("tiny2020"), a few historically important stop lists. This aims to be a transparent and well-document collection of stoplists. Only includes English language stoplists at the moment.

Usage

```
get_stoplist(source = "tiny2020", language = "en", tidy = FALSE)
```

Arguments

`source`	Character indicating source, default = `"tiny2020"`
`language`	Character (default = "en") indicating language of stopwords by ISO 639-1 code, currently only English is supported.
`tidy`	logical (default = `FALSE`), returns a tibble

Details

There is no such thing as a *stopword*! But, there are **tons** of precompiled lists of words that someone thinks we should remove from our texts. (See for example: https://github.com/igorbrigadir/stopwords) One of the first stoplists is from C.J. van Rijsbergen's "Information retrieval: theory and practice" (1979) and includes 250 words. `text2map`'s very own stoplist `tiny2020` is a lean 34 words.

Below are stoplists available with get_stoplist:

- "tiny2020": Tiny (2020) list of 33 words (Default)
- "snowball2001": Snowball stemming package's (2001) list of 127 words
- "snowball2014": Updated Snowball (2014) list of 175 words
- "van1979": C. J. van Rijsbergen's (1979) list of 250 words
- "fox1990": Christopher Fox's (1990) list of 421 words
- "smart1993": Original SMART (1993) list of 570 words
- "onix2000": ONIX (2000) list of 196 words
- "nltk2001": Python's NLTK (2009) list of 179 words

The Snowball (2014) stoplist is likely the most commonly, it is the default in the `stopwords` package, which is used by `quanteda`, `tidytext` and `tokenizers` packages, followed closely by the Smart (1993) stoplist, the default in the `tm` package. The word counts for SMART (1993) and ONIX (2000) are slightly different than in other places because of duplicate words.

Value

Character vector of words to be stopped, if tidy = TRUE, a tibble is returned

Author(s)

Dustin Stoltz

`jfk_speech` *Full Text of JFK's Rice Speech*

Description

This is a data frame for the text of JFK's Rice Speech "We choose to go to the moon." Each row is a 10 word string of the speech – roughly a sentence. This is intended to be used for example code.

Usage

```
jfk_speech
```

Format

A data frame with 2 columns

Variables

Variables:

- sentence_id. Order and unique ID for the sentence
- sentence. The text of a sentence

`meta_shakespeare` *Metadata for Shakespeare's First Folio*

Description

Metadata related to Shakespeare's First Folio including the IDs to download the plays from Project Gutenberg, and a count of the number of deaths in each play (body count).

Usage

```
meta_shakespeare
```

Format

A matrix of 37 rows and 8 columns

Variables

Variables:

- short_title.
- gutenberg_title.
- gutenberg_id.
- genre.
- year.
- body_count.
- boas_problem_plays.
- death.

perm_tester *Monte Carlo Permutation Tests for Model P-Values*

Description

`perm_tester()` carries out Monte Carlo permutation tests for model p-values from two-tailed, left-tailed, and/or right-tailed hypothesis testing.

Usage

```
perm_tester(
  data,
  model,
  perm_var = NULL,
  strat_var = NULL,
  statistic,
  perm_n = 1000,
  alternative = "all",
  alpha = 0.05,
  seed = NULL
)
```

Arguments

`data`	The dataframe from which the model is estimated.
`model`	The model which will be estimated and re-estimated.
`perm_var`	The variable in the model that will be permuted. Default is `NULL` which takes the first `Yn` term in the formula of the model
`strat_var`	Categorical variable for within-stratum permutations. Defaults to `NULL`.
`statistic`	The name of the model statistic you want to "grab" after re-running the model with each permutation to compare to the original model statistic.
`perm_n`	The total number of permutations. Defaults to 1000.
`alternative`	The alternative hypothesis. One of `"two.sided"` (default), `"left"`, `"right"`, and `"all"`. Defaults to `"all"`, which reports the p-value statistics for all three alternative hypotheses.
`alpha`	Alpha level for the hypothesis test. Defaults to 0.05.
`seed`	Optional seed for reproducibility of the p-value statistics. Defaults to null.

Details

`perm_tester()` can be used to derive p-values under the randomization model of inference. There are various reasons one might want to do this— with text data, and observational data more generally, this might be because the corpus/sample is not a random sample from a target population. In such cases, population model p-values might not make much sense since the asymptotically-derived standard errors from which they are constructed themselves do not make sense. We might therefore want to make inferences on the basis of whether or not randomness, as a data-generating mechanism, might reasonably account for a statistic at least as extreme as the one we observed. `perm_tester()` works from this idea.

`perm_tester()` works like this. First, the model (supplied the `model` parameter) is run on the observed data. Second, we take some statistic of interest, which we indicate with the `statistic` parameter, and set it to the side. Third, a variable, `perm_var`, is permuted—meaning the observed values for the rows of `data` on `perm_var` are randomly reshuffled. Fourth, we estimate the model again, this time with the permuted `perm_var`. Fifth, we get grab that same `statistic`. We repeat steps two through five a total of `perm_n` times, each time tallying the number of times the `statistic` from the permutation-derived model is greater than or equal to (for a right-tailed test), less-than or equal to (for a left-tailed test), and/or has an absolute value greater than or equal to (for a two-tailed test) the `statistic` from the "real" model.

If we divide those tallies by the total number of permutations, then we get randomization-based p-values. This is what `perm_tester()` does. The null hypothesis is that randomness could likely generate the statistic that we observe. The alternative hypothesis is that randomness alone likely can't account for the observed statistic.

We then reject the null hypothesis if the p-value is below a threshold indicated with `alpha`, which, as in population-based inference, is the probability below which we are willing to reject the null hypothesis when it is actually true. So if the p-value is below, say, `alpha` = 0.05 and we're performing, a right-tailed test, then fewer than 5% of the statistics derived from the permutation-based models are greater than or equal to our observed statistic. We would then reject the null, as it is unlikely (based on our `alpha` threshold), that randomness as a data-generating mechanism can account for a test statistic at least as large the one we observed.

In most cases, analysts probably cannot expect to perform "exact" permutation tests where every possible permutation is accounted for—i.e., where `perm_n` equals the total number of possible permutations. Instead, we can take random samples of the "population" of permutations. `perm_tester()` does this, and reports the standard errors and (1 - `alpha`) confidence intervals for the p-values.

`perm_tester()` can also perform stratified permutation tests, where the observed `perm_var` variables within groups. This can be done by setting the `strat_var` variable to be he grouping variable.

Value

Returns a data frame with the observed statistic (`stat`), the p-values (`P_left`, for left-tailed, `P_right` for right-tailed, and/or `P_two` for two-tailed), and the standard errors and confidence intervals for those p-values, respectively.

Author(s)

Marshall Taylor and Dustin Stoltz

References

Taylor, Marshall A. (2020) 'Visualization Strategies for Regression Estimates with Randomization Inference' *Stata Journal* 20(2):309-335. doi:10.1177/1536867X20930999.

#' Darlington, Richard B. and Andrew F. Hayes (2016) *Regression analysis and linear models: Concepts, applications, and implementation*. Guilford Publications.

Ernst, Michael D. (2004) 'permutation methods: a basis for exact inference' *Statistical Scicence* 19(4):676-685. doi:10.1214/088342304000000396.

Manly, Bryan F. J. (2007) *Randomization, Bootstrap and Monte Carlo Methods in Biology*. Chapman and Hall/CRC. doi:10.1201/9781315273075.

See Also

CMDist, CoCA, get_direction

Examples

```
data <- text2map::meta_shakespeare

model <- lm(body_count ~ boas_problem_plays + year + genre, data = data)

# without stratified permutations, two-sided test
out1 <- perm_tester(
  data = data,
  model = model,
  statistic = "coefficients",
  perm_n = 40,
  alternative = "two.sided",
  alpha = .01,
  seed = 8675309
)

# with stratified permutations, two-sided test
out2 <- perm_tester(
  data = data,
  model = model,
  strat_var = "boas_problem_plays",
  statistic = "coefficients",
  perm_n = 40,
  alternative = "two.sided",
  alpha = .01,
  seed = 8675309
)
```

`plot.CoCA` *Plot CoCA*

Description

Plot CoCA

Usage

```
## S3 method for class 'CoCA'
plot(
  x,
  module = NULL,
  cutoff = 0.05,
  repulse = 1.86,
  min = 0.15,
  max = 1,
  main = NULL,
  ...
)
```

Arguments

`x`	CoCA object returned by `CoCA()`
`module`	index for which module to plot (default = NULL)
`cutoff`	minimum absolute value of correlations to plot
`repulse`	repulse radius in the spring layout
`min`	edges with absolute weights under this value are not shown (default = 0.15)
`max`	highest weight to scale the edge widths too (default = 1)
`main`	title for plot (default = NULL)
`...`	Arguments to be passed to methods

Value

returns `qgraph` object

`print.CoCA` *Prints CoCA class information*

Description

Prints CoCA class information

Usage

```
## S3 method for class 'CoCA'
print(x, ...)
```

Arguments

`x`	CoCA object returned by `CoCA()`
`...`	Arguments to be passed to methods

Value

prints a message indicating the classes and sizes

`rancors_builder` *Build Multiple Random Corpora*

Description

`rancors_builder()` generates multiple random corpus (rancor) based on a user defined term probabilities and vocabulary. sers can set the number of documents, as well as the mean, standard deviation, minimum, and maximum document lengths (i.e., number of tokens) of the parent normal distribution from which the document lengths are randomly sampled. The output is a list of document-term matrices. To produce a *single* random corpus, use `rancor_builder()` (note the singular).

Usage

```
rancors_builder(
  data,
  vocab,
  probs,
  n_cors,
  n_docs,
  len_mean,
  len_var,
  len_min,
  len_max,
  seed = NULL
)
```

Arguments

`data`	Data.frame containing vocabulary and probabilities
`vocab`	Name of the column containing vocabulary
`probs`	Name of the column containing probabilities
`n_cors`	Integer indicating the number of corpora to build
`n_docs`	Integer(s) indicating the number of documents to be returned If two numbers are provide, number will be randomly sampled within the range for each corpora.
`len_mean`	Integer(s) indicating the mean of the document lengths in the parent normal sampling distribution. If two numbers are provided, number will be randomly sampled within the range for each corpora.
`len_var`	Integer(s) indicating the standard deviation of the document lengths in the parent normal sampling distribution. If two numbers are provided, number will be randomly sampled within the range for each corpora.
`len_min`	Integer(s) indicating the minimum of the document lengths in the parent normal sampling distribution. If two numbers are provided, number will be randomly sampled within the range for each corpora.
`len_max`	Integer(s) indicating the maximum of the document lengths in the parent normal sampling distribution. If two numbers are provided, number will be randomly sampled within the range for each corpora.
`seed`	Optional seed for reproducibility

Author(s)

Dustin Stoltz and Marshall Taylor

Examples

```
# create corpus and DTM
my_corpus <- data.frame(
  text = c(
    "I hear babies crying I watch them grow",
    "They'll learn much more than I'll ever know",
    "And I think to myself",
    "What a wonderful world",
    "Yes I think to myself",
    "What a wonderful world"
  ),
  line_id = paste0("line", seq_len(6))
)
## some text preprocessing
my_corpus$clean_text <- tolower(gsub("'", "", my_corpus$text))

dtm <- dtm_builder(
  data = my_corpus,
  text = clean_text,
  doc_id = line_id
)

# use colSums to get term frequencies
df <- data.frame(
  vocab = colnames(dtm),
  freqs = colSums(dtm)
)
# convert to probabilities
df$probs <- df$freqs / sum(df$freqs)

# create random DTM
ls_dtms <- df |>
  rancors_builder(vocab,
    probs,
    n_cors = 20,
    n_docs = 100,
    len_mean = c(50, 200),
    len_var = 5,
    len_min = 20,
    len_max = 1000,
    seed = 59801
  )
length(ls_dtms)
```

---

rancor_builder *Build a Random Corpus*

Description

rancor_builder() generates a random corpus (rancor) based on a user defined term probabilities and vocabulary. Users can set the number of documents, as well as the mean, standard deviation, minimum, and maximum document lengths (i.e., number of tokens) of the parent normal distribution from which the document lengths are randomly sampled. The output is a single document-term matrix. To produce multiple random corpora, use rancors_builder() (note the plural). Term probabilities/vocabulary can come from a users own corpus, or a pre-compiled frequency list, such as the one derived from the Google Book N-grams corpus

Usage

```
rancor_builder(
  data,
  vocab,
  probs,
  n_docs = 100L,
  len_mean = 500,
  len_var = 10L,
  len_min = 20L,
  len_max = 1000L,
  seed = NULL
)
```

Arguments

data	Data.frame containing vocabulary and probabilities
vocab	Name of the column containing vocabulary
probs	Name of the column containing probabilities
n_docs	Integer indicating the number of documents to be returned
len_mean	Integer indicating the mean of the document lengths in the parent normal sampling distribution
len_var	Integer indicating the standard deviation of the document lengths in the parent normal sampling distribution
len_min	Integer indicating the minimum of the document lengths in the parent normal sampling distribution
len_max	Integer indicating the maximum of the document lengths in the parent normal sampling distribution
seed	Optional seed for reproducibility

Author(s)

Dustin Stoltz and Marshall Taylor

Examples

```
# create corpus and DTM
my_corpus <- data.frame(
  text = c(
    "I hear babies crying I watch them grow",
    "They'll learn much more than I'll ever know",
    "And I think to myself",
    "What a wonderful world",
    "Yes I think to myself",
    "What a wonderful world"
  ),
  line_id = paste0("line", seq_len(6))
)
## some text preprocessing
my_corpus$clean_text <- tolower(gsub("'", "", my_corpus$text))

dtm <- dtm_builder(
  data = my_corpus,
  text = clean_text,
  doc_id = line_id
)

# use colSums to get term frequencies
df <- data.frame(
  terms = colnames(dtm),
  freqs = colSums(dtm)
)
# convert to probabilities
df$probs <- df$freqs / sum(df$freqs)

# create random DTM
rDTM <- df |>
  rancor_builder(terms, probs)
```

`seq_builder` *Represent Documents as Token-Integer Sequences*

Description

First, each token in the vocabulary is mapped to an integer in a lookup dictionary. Next, documents are converted to sequences of integers where each integer is an index of the token from the dictionary.

Usage

```
seq_builder(
  data,
  text,
  doc_id = NULL,
  vocab = NULL,
  maxlen = NULL,
  matrix = TRUE
)
```

Arguments

`data`	Data.frame with column of texts and column of document ids
`text`	Name of the column with documents' text
`doc_id`	Name of the column with documents' unique ids.
`vocab`	Default is `NULL`, if a list of terms is provided, the function will return a DTM with terms restricted to this vocabulary. Columns will also be in the same order as the list of terms.
`maxlen`	Integer indicating the maximum document length. If NULL (default), the length of the longest document is used.
`matrix`	Logical, `TRUE` (default) returns a matrix, `FALSE` a list

Details

Function will return a matrix of integer sequences by default. The columns will be the length of the longest document or `maxlen`, with shorter documents padded with zeros. The dictionary will be an attribute of the matrix accessed with `attr(seq, "dic")`. If `matrix = FALSE`, the function will return a list of integer sequences. The vocabulary will either be each unique token in the corpus, or a the list of words provided to the `vocab` argument. This kind of text representation is used in tensorflow and keras.

Value

returns a matrix or list

Author(s)

Dustin Stoltz

`stoplists` *A dataset of stoplists*

Description

A dataset containing eight English stoplist. Is used with the `get_stoplist()` function.

Usage

```
stoplists
```

Format

A data frame with 1775 rows and 2 variables.

Details

The stoplists include:

- "tiny2020": Tiny (2020) list of 33 words (Default)
- "snowball2001": Snowball (2001) list of 127 words
- "snowball2014": Updated Snowball (2014) list of 175 words
- "van1979": van Rijsbergen's (1979) list of 250 words
- "fox1990": Christopher Fox's (1990) list of 421 words
- "smart1993": Original SMART (1993) list of 570 words
- "onix2000": ONIX (2000) list of 196 words
- "nltk2001": Python's NLTK (2009) list of 179 words

Tiny 2020, is a very small stop list of the most frequent English conjunctions, articles, prepositions, and demonstratives (N=17). Also includes the 8 forms of the copular verb "to be" and the 8 most frequent personal (singular and plural) pronouns (minus gendered and possessive pronouns).

No contractions are included.

Variables

Variables:

- words. words to be stopped
- source. source of the list

`test_anchors` *Evaluate anchor sets in defining semantic directions*

Description

This function evaluates how well an anchor set defines a semantic direction. Anchors must be a two-column data.frame or a list of length == 2. Currently, the function only implements the "PairDir" metric developed by Boutyline and Johnston (2023).

Usage

```
test_anchors(anchors, wv, method = c("pairdir"), all = FALSE, summarize = TRUE)
```

Arguments

`anchors`	A data frame or list of juxtaposed 'anchor' terms
`wv`	Matrix of word embedding vectors (a.k.a embedding model) with rows as terms.
`method`	Which metric used to evaluate (currently only pairdir)
`all`	Logical (default `FALSE`). Whether to evaluate all possible pairwise combinations of two sets of anchors. If `FALSE` only the input pairs are used in evaluation and anchor sets must be of equal lengths.
`summarize`	Logical (default `TRUE`). Returns a dataframe with AVERAGE scores for input pairs along with each pairs' contribution. If `summarize = FALSE`, returns a list with each offset matrix, each contribution, and the average score.

Details

According to Boutyline and Johnston (2023):

"We find that PairDir – a measure of parallelism between the offset vectors (and thus of the internal reliability of the estimated relation) – consistently outperforms other reliability metrics in explaining axis accuracy."

Boutyline and Johnston only consider analyst specified pairs. However, if `all = TRUE`, all pairwise combinations of terms between each set are evaluated. This can allow for unequal sets of anchors, however this increases computational complexity considerably.

Value

dataframe or list

References

Boutyline, Andrei, and Ethan Johnston. 2023. “Forging Better Axes: Evaluating and Improving the Measurement of Semantic Dimensions in Word Embeddings.” doi:10.31235/osf.io/576h3

Examples

```
# load example word embeddings
data(ft_wv_sample)

df_anchors <- data.frame(
  a = c("rest", "rested", "stay", "stand"),
  z = c("coming", "embarked", "fast", "move")
)

test_anchors(df_anchors, ft_wv_sample)

test_anchors(df_anchors, ft_wv_sample, all = TRUE)
```

`tiny_gender_tagger` *A very tiny "gender" tagger*

Description

Provides a small dictionary which matches common English pronouns and nouns to conventional gender categories ("masculine" or "feminine"). There are 20 words in each category.

Usage

```
tiny_gender_tagger()
```

Value

returns a tibble with two columns

Author(s)

Dustin Stoltz

`vocab_builder` *A fast unigram vocabulary builder*

Description

A streamlined function to take raw texts from a column of a data.frame and produce a list of all the unique tokens. Tokenizes by the fixed, single whitespace, and then extracts the unique tokens. This can be used as input to `dtm_builder()` to standardize the vocabulary (i.e. the columns) across multiple DTMs. Prior to building the vocabulary, texts should have whitespace trimmed, if desired, punctuation removed and terms lowercased.

Usage

```
vocab_builder(data, text)
```

Arguments

`data`	Data.frame with one column of texts
`text`	Name of the column with documents' text

Value

returns a list of unique terms in a corpus

Author(s)

Dustin Stoltz

Index