

Package ‘text2vec’

August 8, 2017

Type Package

Version 0.5.0

Date 2017-08-07

Title Modern Text Mining Framework for R

License GPL (>= 2) | file LICENSE

Description Fast and memory-friendly tools for text vectorization, topic modeling (LDA, LSA), word embeddings (GloVe), similarities. This package provides a source-agnostic streaming API, which allows researchers to perform analysis of collections of documents which are larger than available RAM. All core functions are parallelized to benefit from multicore machines.

Maintainer Dmitriy Selivanov <selivanov.dmitriy@gmail.com>

Encoding UTF-8

SystemRequirements GNU make, C++11

Depends R (>= 3.2.0), methods

Imports Matrix (>= 1.1), Rcpp (>= 0.11), RcppParallel (>= 4.3.14), digest (>= 0.6.8), foreach (>= 1.4.3), data.table (>= 1.9.6), magrittr (>= 1.5), irlba (>= 2.2.1), R6 (>= 2.1.2), uuid (>= 0.1-2), futile.logger (>= 1.4.3)

LinkingTo Rcpp, RcppParallel, digest, sparsepp (>= 0.2.0)

Suggests stringr (>= 1.1.0), doParallel, testthat, covr, knitr, rmarkdown, glmnet, parallel

URL <http://text2vec.org>

BugReports <https://github.com/dselivanov/text2vec/issues>

VignetteBuilder knitr

LazyData true

RoxygenNote 6.0.1

NeedsCompilation yes

Author Dmitriy Selivanov [aut, cre, cph],
Qing Wang [aut, cph] (Author of the WapRLDA C++ code)

Repository CRAN

Date/Publication 2017-08-08 04:43:48 UTC

R topics documented:

as.lda_c	2
check_analogy_accuracy	3
collect	3
Collocations	4
create_dtm	6
create_tcm	7
create_vocabulary	9
distances	11
fit	12
fit_transform	12
GlobalVectors	13
glove	15
ifiles	16
itoken	17
LatentDirichletAllocation	19
LatentSemanticAnalysis	21
movie_review	22
normalize	23
perplexity	24
prepare_analogy_questions	25
prune_vocabulary	25
RelaxedWordMoversDistance	26
similarities	28
split_into	28
text2vec	29
TfIdf	29
tokenizers	30
transform	31
vectorizers	32
Index	34

as.lda_c	<i>Converts document-term matrix sparse matrix to 'lda_c' format</i>
----------	--

Description

Converts 'dgCMatrix' (or coercible to 'dgCMatrix') to 'lda_c' format

Usage

```
as.lda_c(X)
```

Arguments

X	Document-Term matrix
---	----------------------

`check_analogy_accuracy`*Checks accuracy of word embeddings on the analogy task*

Description

This function checks how well the GloVe word embeddings do on the analogy task. For full examples see [glove](#).

Usage

```
check_analogy_accuracy(questions_list, m_word_vectors)
```

Arguments

`questions_list` list of questions. Each element of `questions_list` is a integer matrix with four columns. It represents a set of questions related to a particular category. Each element of matrix is an index of a row in `m_word_vectors`. See output of [prepare_analogy_questions](#) for details

`m_word_vectors` word vectors numeric matrix. Each row should represent a word.

See Also

[prepare_analogy_questions](#), [glove](#)

`collect`*Collects distributed data structure to master process*

Description

Collects distributed data structure to master process

Usage

```
collect(x, ...)
```

```
## S3 method for class 'RowDistributedMatrix'
```

```
collect(x, ...)
```

Arguments

`x` pointer to distributed data structure (only row-distributed matrices at the moment)

`...` other options (not used at the moment)

 Collocations

Collocations model.

Description

Creates Collocations model which can be used for phrase extraction.

Usage

Collocations

Format

[R6Class](#) object.

Fields

`collocation_stat` data.table with `collocations(phrases)` statistics. Useful for filtering non-relevant phrases

Usage

For usage details see **Methods, Arguments and Examples** sections.

```
colloc = Collocations$new(vocabulary = NULL, collocation_count_min = 50, pmi_min = 5, gensim_min = 0,
  colloc$partial_fit(it, ...)
  colloc$fit(it, n_iter = 1, ...)
  colloc$transform(it)
  colloc$prune(pmi_min = 5, gensim_min = 0, lfmd_min = -Inf)
  colloc$collocation_stat
```

Methods

`$new(vocabulary = NULL, collocation_count_min = 50, sep = "_")` Constructor for Collocations model. For description of arguments see **Arguments** section.

`$fit(it, n_iter = 1, ...)` fit Collocations model to input iterator `it`. Iterating over input iterator `it` `n_iter` times, so hierarchically can learn multi-word phrases. Invisibly returns `collocation_stat`.

`$partial_fit(it, ...)` iterates once over data and learns collocations. Invisibly returns `collocation_stat`. Workhorse for `$fit()`.

`$transform(it)` transforms input iterator using learned collocations model. Result of the transformation is new `itoken` or `itoken_parallel` iterator which will produce tokens with phrases collapsed into single token.

`$prune(pmi_min = 5, gensim_min = 0, lfmd_min = -Inf)` filter out non-relevant phrases with low score. User can do it directly by modifying `collocation_stat` object.

Arguments

colloc A Collocation model object

n_iter number of iteration over data

pmi_min, gensim_min, lfmd_min minimal scores of the corresponding statistics in order to collapse tokens into collocation see data in `colloc$collocation_stat` for better understanding

it An input itoken or itoken_parallel iterator

vocabulary `text2vec_vocabulary` - if provided will look for collocations consisted of only from vocabulary

Examples

```
library(text2vec)
data("movie_review")

preprocessor = function(x) {
  tolower(x) %>% gsub("[^[:alnum:]]\\s", replacement = " ", .)
}
sample_ind = 1:100
tokens = movie_review$review[sample_ind] %>% preprocessor %>% word_tokenizer
it = itoken(tokens, ids = movie_review$id[sample_ind])
system.time(v <- create_vocabulary(it))
v = prune_vocabulary(v, term_count_min = 5)

cc = Collocations$new(collocation_count_min = 5, pmi_min = 5)
cc$fit(it, n_iter = 2)
cc$collocation_stat

it2 = cc$transform(it)
v2 = create_vocabulary(it2)
v2 = prune_vocabulary(v2, term_count_min = 5)
# check what phrases model has learned
setdiff(v2$term, v$term)
# [1] "main_character" "jeroen_krabb" "boogey_man" "in_order"
# [5] "couldn_t" "much_more" "my_favorite" "worst_film"
# [9] "have_seen" "characters_are" "i_mean" "better_than"
# [13] "don_t_care" "more_than" "look_at" "they_re"
# [17] "each_other" "must_be" "sexual_scenes" "have_been"
# [21] "there_are_some" "you_re" "would_have" "i_loved"
# [25] "special_effects" "hit_man" "those_who" "people_who"
# [29] "i_am" "there_are" "could_have_been" "we_re"
# [33] "so_bad" "should_be" "at_least" "can_t"
# [37] "i_thought" "isn_t" "i_ve" "if_you"
# [41] "didn_t" "doesn_t" "i_m" "don_t"

# and same way we can create document-term matrix which contains
# words and phrases!
dtm = create_dtm(it2, vocab_vectorizer(v2))
# check that dtm contains phrases
which(colnames(dtm) == "jeroen_krabb")
```

`create_dtm`*Document-term matrix construction*

Description

This is a high-level function for creating a document-term matrix.

Usage

```
create_dtm(it, vectorizer, type = c("dgCMatrix", "dgTMatrix"), ...)  
  
## S3 method for class 'itoken'  
create_dtm(it, vectorizer, type = c("dgCMatrix",  
  "dgTMatrix"), ...)  
  
## S3 method for class 'list'  
create_dtm(it, vectorizer, type = c("dgCMatrix", "dgTMatrix"),  
  ...)  
  
## S3 method for class 'itoken_parallel'  
create_dtm(it, vectorizer, type = c("dgCMatrix",  
  "dgTMatrix"), distributed = FALSE, ...)
```

Arguments

<code>it</code>	<code>itoken</code> iterator or list of <code>itoken</code> iterators.
<code>vectorizer</code>	function <code>vectorizer</code> function; see vectorizers .
<code>type</code>	character, one of <code>c("dgCMatrix", "dgTMatrix")</code> .
<code>...</code>	arguments to the foreach function which is used to iterate over <code>it</code> .
<code>distributed</code>	whether to keep matrix row-distributed. Row-distributed matrices can be used in distributed LDA.

Details

If a parallel backend is registered and first argument is a list of `itoken`, iterators, function will construct the DTM in multiple threads. User should keep in mind that he or she should split the data itself and provide a list of `itoken` iterators. Each element of `it` will be handled in separate thread and combined at the end of processing.

Value

A document-term matrix

See Also

[itoken vectorizers](#)

Examples

```

## Not run:
data("movie_review")
N = 1000
it = itoken(movie_review$review[1:N], preprocess_function = tolower,
            tokenizer = word_tokenizer)
v = create_vocabulary(it)
#remove very common and uncommon words
pruned_vocab = prune_vocabulary(v, term_count_min = 10,
                                doc_proportion_max = 0.5, doc_proportion_min = 0.001)
vectorizer = vocab_vectorizer(v)
it = itoken(movie_review$review[1:N], preprocess_function = tolower,
            tokenizer = word_tokenizer)
dtm = create_dtm(it, vectorizer)
# get tf-idf matrix from bag-of-words matrix
dtm_tfidf = transformer_tfidf(dtm)

## Example of parallel mode
# set to number of cores on your machine
N_WORKERS = 1
if(require(doParallel)) registerDoParallel(N_WORKERS)
splits = split_into(movie_review$review, N_WORKERS)
jobs = lapply(splits, itoken, tolower, word_tokenizer, n_chunks = 1)
vectorizer = hash_vectorizer()
dtm = create_dtm(jobs, vectorizer, type = 'dgTMatrix')

## End(Not run)

```

create_tcm

Term-co-occurrence matrix construction

Description

This is a function for constructing a term-co-occurrence matrix(TCM). TCM matrix usually used with [GloVe](#) word embedding model.

Usage

```

create_tcm(it, vectorizer, skip_grams_window = 5L,
           skip_grams_window_context = c("symmetric", "right", "left"),
           weights = 1/seq_len(skip_grams_window), ...)

## S3 method for class 'itoken'
create_tcm(it, vectorizer, skip_grams_window = 5L,
           skip_grams_window_context = c("symmetric", "right", "left"),
           weights = 1/seq_len(skip_grams_window), ...)

## S3 method for class 'itoken_parallel'
create_tcm(it, vectorizer, skip_grams_window = 5L,

```

```
skip_grams_window_context = c("symmetric", "right", "left"),
weights = 1/seq_len(skip_grams_window), ...)
```

Arguments

<code>it</code>	list of iterators over tokens from itoken . Each element is a list of tokens, that is, tokenized and normalized strings.
<code>vectorizer</code>	function vectorizer function. See vectorizers .
<code>skip_grams_window</code>	integer window for term-co-occurrence matrix construction. <code>skip_grams_window</code> should be > 0 if you plan to use vectorizer in create_tcm function. Value of <code>0L</code> means to not construct the TCM.
<code>skip_grams_window_context</code>	one of <code>c("symmetric", "right", "left")</code> - which context words to use when count co-occurrence statistics.
<code>weights</code>	weights for context/distant words during co-occurrence statistics calculation. By default we are setting <code>weight = 1 / distance_from_current_word</code> . Should have length equal to <code>skip_grams_window</code> . "symmetric" by default - take into account <code>skip_grams_window</code> left and right.
<code>...</code>	arguments to foreach function which is used to iterate over it.

Details

If a parallel backend is registered, it will construct the TCM in multiple threads. The user should keep in mind that he/she should split data and provide a list of [itoken](#) iterators. Each element of it will be handled in a separate thread combined at the end of processing.

Value

`dgTMatrix` TCM matrix

See Also

[itoken](#) [create_dtm](#)

Examples

```
## Not run:
data("movie_review")

# single thread

tokens = movie_review$review %>% tolower %>% word_tokenizer
it = itoken(tokens)
v = create_vocabulary(jobs)
vectorizer = vocab_vectorizer(v)
tcm = create_tcm(itoken(tokens), vectorizer, skip_grams_window = 3L)

# parallel version
```



```

# set to number of cores on your machine
N_WORKERS = 1
if(require(doParallel)) registerDoParallel(N_WORKERS)
splits = split_into(movie_review$review, N_WORKERS)
jobs = lapply(splits, itoken, tolower, word_tokenizer)
v = create_vocabulary(jobs)
vectorizer = vocab_vectorizer(v)
jobs = lapply(splits, itoken, tolower, word_tokenizer)

tcm = create_tcm(jobs, vectorizer, skip_grams_window = 3L, skip_grams_window_context = "symmetric")

## End(Not run)

```

create_vocabulary *Creates a vocabulary of unique terms*

Description

This function collects unique terms and corresponding statistics. See the below for details.

Usage

```

create_vocabulary(it, ngram = c(ngram_min = 1L, ngram_max = 1L),
  stopwords = character(0), sep_ngram = "_")

vocabulary(it, ngram = c(ngram_min = 1L, ngram_max = 1L),
  stopwords = character(0), sep_ngram = "_")

## S3 method for class 'character'
create_vocabulary(it, ngram = c(ngram_min = 1L, ngram_max
  = 1L), stopwords = character(0), sep_ngram = "_")

## S3 method for class 'itoken'
create_vocabulary(it, ngram = c(ngram_min = 1L, ngram_max =
  1L), stopwords = character(0), sep_ngram = "_")

## S3 method for class 'list'
create_vocabulary(it, ngram = c(ngram_min = 1L, ngram_max =
  1L), stopwords = character(0), sep_ngram = "_", ...)

## S3 method for class 'itoken_parallel'
create_vocabulary(it, ngram = c(ngram_min = 1L,
  ngram_max = 1L), stopwords = character(0), sep_ngram = "_", ...)

```

Arguments

it iterator over a list of character vectors, which are the documents from which the user wants to construct a vocabulary. See [itoken](#). Alternatively, a character vector of user-defined vocabulary terms (which will be used "as is").

ngram	integer vector. The lower and upper boundary of the range of n-values for different n-grams to be extracted. All values of n such that <code>ngram_min <= n <= ngram_max</code> will be used.
stopwords	character vector of stopwords to filter out
sep_ngram	character a character string to concatenate words in ngrams
...	additional arguments to <code>foreach</code> function.

Value

text2vec_vocabulary object, which is actually a data.frame with following columns:

term	character vector of unique terms
term_count	integer vector of term counts across all documents
doc_count	integer vector of document counts that contain corresponding term

Also it contains metainformation in attributes: `ngram`: integer vector, the lower and upper boundary of the range of n-gram-values. `document_count`: integer number of documents vocabulary was built. `stopwords`: character vector of stopwords `sep_ngram`: character separator for ngrams

Methods (by class)

- `character`: creates `text2vec_vocabulary` from predefined character vector. Terms will be inserted **as is**, without any checks (ngrams number, ngram delimiters, etc.).
- `itoken`: collects unique terms and corresponding statistics from object.
- `list`: collects unique terms and corresponding statistics from list of `itoken` iterators. If parallel backend is registered, it will build vocabulary in parallel using `foreach`.
- `itoken_parallel`: collects unique terms and corresponding statistics from iterator. If parallel backend is registered, it will build vocabulary in parallel using `foreach`.

Examples

```
data("movie_review")
txt = movie_review[['review']][1:100]
it = itoken(txt, tolower, word_tokenizer, n_chunks = 10)
vocab = create_vocabulary(it)
pruned_vocab = prune_vocabulary(vocab, term_count_min = 10, doc_proportion_max = 0.8,
doc_proportion_min = 0.001, vocab_term_max = 20000)
```

distances

Pairwise Distance Matrix Computation

Description

`dist2` calculates pairwise distances/similarities between the rows of two data matrices. **Note** that some methods work only on sparse matrices and others work only on dense matrices.

`pdist2` calculates "parallel" distances between the rows of two data matrices.

Usage

```
dist2(x, y = NULL, method = c("cosine", "euclidean", "jaccard"),
      norm = c("l2", "l1", "none"))
```

```
pdist2(x, y, method = c("cosine", "euclidean", "jaccard"), norm = c("l2",
"l1", "none"))
```

Arguments

<code>x</code>	first matrix.
<code>y</code>	second matrix. For <code>dist2</code> <code>y = NULL</code> set by default. This means that we will assume <code>y = x</code> and calculate distances/similarities between all rows of the <code>x</code> .
<code>method</code>	usually character or instance of <code>tet2vec_distance</code> class. The distances/similarity measure to be used. One of <code>c("cosine", "euclidean", "jaccard")</code> or RWMD . RWMD works only on bag-of-words matrices. In case of "cosine" distance max distance will be 1 - (-1) = 2
<code>norm</code>	character = <code>c("l2", "l1", "none")</code> - how to scale input matrices. If they already scaled - use "none"

Details

Computes the distance matrix computed by using the specified method. Similar to [dist](#) function, but works with two matrices.

`pdist2` takes two matrices and return a single vector. giving the 'parallel' distances of the vectors.

Value

`dist2` returns matrix of distances/similarities between each row of matrix `x` and each row of matrix `y`.

`pdist2` returns vector of "parallel" distances between rows of `x` and `y`.

fit	<i>Fits model to data</i>
-----	---------------------------

Description

Generic function to fit models - inherited from estimator

Usage

```
fit(x, model, y = NULL, ...)

## S3 method for class 'Matrix'
fit(x, model, y = NULL, ...)

## S3 method for class 'matrix'
fit(x, model, y = NULL, ...)
```

Arguments

x	a matrix like object, should inherit from <code>Matrix</code> or <code>matrix</code>
model	instance of class <code>estimator</code> which should implement method with signature <code>\$fit(x, y, ...)</code>
y	NULL by default. Optional response variable for supervised learning models. Should inherit from vector or <code>Matrix</code> or <code>matrix</code> . See documentation for corresponding models.
...	additional data/model dependent arguments to downstream functions.

Value

```
invisible(object$self())
```

fit_transform	<i>Fit model to data, then transform it</i>
---------------	---

Description

This is generic function to fit transformers (`class = "transformer"`) and then apply fitted model to input.

Usage

```
fit_transform(x, model, y = NULL, ...)

## S3 method for class 'Matrix'
fit_transform(x, model, y = NULL, ...)

## S3 method for class 'matrix'
fit_transform(x, model, y = NULL, ...)
```

Arguments

x	a matrix like object, should inherit from Matrix or matrix
model	instance of class estimator which should implement method with signature \$fit(x, ...)
y	NULL by default. Optional response variable for supervised models. Should inherit from vector Matrix or matrix. See documentation for corresponding models.
...	additional data/model dependent arguments to downstream functions.

Value

Transformed version of x

GlobalVectors

Creates Global Vectors word-embeddings model.

Description

Class for GloVe word-embeddings model. It can be trained via fully can asynchronous and parallel AdaGrad with \$fit_transform() method.

Usage

GloVe

Format

[R6Class](#) object.

Fields

components represents context word vectors

n_dump_every integer = 0L by default. Defines frequency of dumping word vectors. For example user can ask to dump word vectors each 5 iteration.

`shuffle` logical = FALSE by default. Defines shuffling before each SGD iteration. Generally shuffling is a good idea for stochastic-gradient descent, but from my experience in this particular case it does not improve convergence.

`grain_size` integer = 1e5L by default. This is the `grain_size` for `RcppParallel::parallelReduce`. For details, see <http://rcppcore.github.io/RcppParallel/#grain-size>. **We don't recommend to change this parameter.**

Usage

For usage details see **Methods, Arguments and Examples** sections.

```
glove = GlobalVectors$new(word_vectors_size, vocabulary, x_max, learning_rate = 0.15,
                          alpha = 0.75, lambda = 0.0, shuffle = FALSE, initial = NULL)
glove$fit_transform(x, n_iter = 10L, convergence_tol = -1, n_check_convergence = 1L,
                   n_threads = RcppParallel::defaultNumThreads(), ...)
glove$components
glove$dump()
```

Methods

`$new(word_vectors_size, vocabulary, x_max, learning_rate = 0.15, alpha = 0.75, lambda = 0, shuffle = FALSE, initial = NULL)`
 Constructor for Global vectors model. For description of arguments see **Arguments** section.

`$fit_transform(x, n_iter = 10L, convergence_tol = -1, n_check_convergence = 1L, n_threads = RcppParallel::defaultNumThreads(), ...)`
 fit GloVe model to input matrix x

`$dump()` get model internals - word vectors and biases for main and context words

`$get_history` get history of SGD costs and word vectors (if `n_dump_every > 0`)

Arguments

glove A GloVe object

x An input term co-occurrence matrix. Preferably in `dgTMatrix` format

n_iter integer number of SGD iterations

word_vectors_size desired dimension for word vectors

vocabulary character vector or instance of `text2vec_vocabulary` class. Each word should correspond to dimension of co-occurrence matrix.

x_max integer maximum number of co-occurrences to use in the weighting function. see the GloVe paper for details: <http://nlp.stanford.edu/pubs/glove.pdf>

learning_rate numeric learning rate for SGD. I do not recommend that you modify this parameter, since AdaGrad will quickly adjust it to optimal

convergence_tol numeric = -1 defines early stopping strategy. We stop fitting when one of two following conditions will be satisfied: (a) we have used all iterations, or (b) `cost_previous_iter / cost_current_iter < convergence_tol`. By default perform all iterations.

alpha numeric = 0.75 the alpha in weighting function formula: $f(x) = 1 \text{ if } x > x_{max}; \text{ else } (x/x_{max})^{\alpha}$

lambda numeric = 0.0, L1 regularization coefficient. 0 = vanilla GloVe, corresponds to original paper and implementation. lambda >0 corresponds to text2vec new feature and different SGD algorithm. From our experience small lambda (like lambda = 1e-5) usually produces better results than vanilla GloVe on small corpora

initial NULL - word vectors and word biases will be initialized randomly. Or named list which contains w_i, w_j, b_i, b_j values - initial word vectors and biases. This is useful for fine-tuning. For example one can pretrain model on large corpus (such as wikipedia dump) and then fine tune on smaller task-specific dataset

See Also

<http://nlp.stanford.edu/projects/glove/>

Examples

```
## Not run:
temp = tempfile()
download.file('http://matmahoney.net/dc/text8.zip', temp)
text8 = readLines(unz(temp, "text8"))
it = itoken(text8)
vocabulary = create_vocabulary(it) %>%
  prune_vocabulary(term_count_min = 5)
v_vect = vocab_vectorizer(vocabulary)
tcm = create_tcm(it, v_vect, skip_grams_window = 5L)
glove_model = GloVe$new(word_vectors_size = 50,
  vocabulary = vocabulary, x_max = 10, learning_rate = .25)
# fit model and get word vectors
word_vectors_main = glove_model$fit_transform(tcm, n_iter = 10)
word_vectors_context = glove_model$components
word_vectors = word_vectors_main + t(word_vectors_context)

## End(Not run)
```

glove

Fit a GloVe word-embedded model

Description

DEPRECATED. This function trains a GloVe word-embeddings model via fully asynchronous and parallel AdaGrad.

Usage

```
glove(tcm, vocabulary_size = nrow(tcm), word_vectors_size, x_max, num_iters,
  shuffle_seed = NA_integer_, learning_rate = 0.05,
  convergence_threshold = -1, grain_size = 100000L, alpha = 0.75, ...)
```

Arguments

tcm	an object which represents a term-co-occurrence matrix, which is used in training. At the moment only dgTMatrix or objects coercible to a dgTMatrix) are supported. In future releases we will add support for out-of-core learning and streaming a TCM from disk.
vocabulary_size	number of words in in the term-co-occurrence matrix
word_vectors_size	desired dimension for word vectors
x_max	maximum number of co-occurrences to use in the weighting function. See the GloVe paper for details: http://nlp.stanford.edu/pubs/glove.pdf .
num_iters	number of AdaGrad epochs
shuffle_seed	integer seed. Use NA_integer_ to turn shuffling off. A seed defines shuffling before each SGD iteration. Parameter only controls shuffling before each SGD iteration. Result still will be unpredictable (because of Hogwild style async SGD)! Generally shuffling is a good idea for stochastic-gradient descent, but from my experience in this particular case it does not improve convergence. By default there is no shuffling. Please report if you find that shuffling improves your score.
learning_rate	learning rate for SGD. I do not recommend that you modify this parameter, since AdaGrad will quickly adjust it to optimal.
convergence_threshold	defines early stopping strategy. We stop fitting when one of two following conditions will be satisfied: (a) we have used all iterations, or (b) $\text{cost_previous_iter} / \text{cost_current_iter} < \text{convergence_threshold}$.
grain_size	I do not recommend adjusting this parameter. This is the grain_size for RcppParallel::parallelReduce. For details, see http://rcppcore.github.io/RcppParallel/#grain-size .
alpha	the alpha in weighting function formula: $f(x) = 1 \text{ if } x > x_{max}; \text{ else } (x/x_{max})^{\alpha}$
...	arguments passed to other methods (not used at the moment).

ifiles	<i>Creates iterator over text files from the disk</i>
--------	---

Description

The result of this function usually used in an `itoken` function.

Usage

```
ifiles(file_paths, reader = readLines)

idir(path, reader = readLines)

ifiles_parallel(file_paths, reader = readLines,
  n_chunks = foreach::getDoParWorkers())
```


Arguments

file_paths	character paths of input files
reader	function which will perform reading of text files from disk, which should take a path as its first argument. reader() function should return named character vector: elements of vector = documents, names of the elements = document ids which will be used in DTM construction . If user doesn't provide named character vector, document ids will be generated as file_name + line_number (assuming that each line is a document).
path	character path of directory. All files in the directory will be read.
n_chunks	integer, defines in how many chunks files will be processed. For example if you have 32 files, and n_chunks = 8, then for each 4 files will be created a job (for example document-term matrix construction). In case some parallel backend registered, each job will be evaluated in a separated thread (process) in parallel. So each such group of files will be processed in parallel and at the end all 8 results from will be combined.

See Also

[itoken](#)

Examples

```
## Not run:
current_dir_files = list.files(path = ".", full.names = TRUE)
files_iterator = ifiles(current_dir_files)
parallel_files_iterator = ifiles_parallel(current_dir_files, n_chunks = 4)
it = itoken_parallel(parallel_files_iterator)
dtm = create_dtm(it, hash_vectorizer(2**16), type = 'dgTMatrix')

## End(Not run)
dir_files_iterator = idir(path = ".")
```

itoken

Iterators (and parallel iterators) over input objects

Description

This family of function creates iterators over input objects in order to create vocabularies, or DTM and TCM matrices. iterators usually used in following functions : [create_vocabulary](#), [create_dtm](#), [vectorizers](#), [create_tcm](#). See them for details.

Usage

```
itoken(iterable, ...)

## S3 method for class 'list'
itoken(iterable, n_chunks = 10, progressbar = interactive(),
```

```

    ids = NULL, ...)

## S3 method for class 'character'
itoken(iterable, preprocessor = identity,
       tokenizer = space_tokenizer, n_chunks = 10, progressbar = interactive(),
       ids = NULL, ...)

## S3 method for class 'iterator'
itoken(iterable, preprocessor = identity,
       tokenizer = space_tokenizer, n_chunks = 1L, progressbar = interactive(),
       ...)

itoken_parallel(iterable, ...)

## S3 method for class 'character'
itoken_parallel(iterable, preprocessor = identity,
               tokenizer = space_tokenizer, n_chunks = foreach::getDoParWorkers(),
               ids = NULL, ...)

## S3 method for class 'ifiles_parallel'
itoken_parallel(iterable, preprocessor = identity,
               tokenizer = space_tokenizer, n_chunks = 1L, ...)

```

Arguments

<code>iterable</code>	an object from which to generate an iterator
<code>...</code>	arguments passed to other methods
<code>n_chunks</code>	integer, the number of pieces that object should be divided into. Then each chunk is processed independently (and in case <code>itoken_parallel</code> in parallel if some parallel backend is registered). Usually there is tradeoff: larger number of chunks means lower memory footprint, but slower (if preprocessor, tokenizer functions are efficiently vectorized). And small number of chunks means larger memory footprint but faster execution (again if user supplied preprocessor, tokenizer functions are efficiently vectorized).
<code>progressbar</code>	logical indicates whether to show progress bar.
<code>ids</code>	vector of document ids. If <code>ids</code> is not provided, <code>names(iterable)</code> will be used. If <code>names(iterable) == NULL</code> , incremental ids will be assigned.
<code>preprocessor</code>	function which takes chunk of character vectors and does all pre-processing. Usually preprocessor should return a character vector of preprocessed/cleaned documents. See "Details" section.
<code>tokenizer</code>	function which takes a character vector from preprocessor, split it into tokens and returns a list of character vectors. If you need to perform stemming - call stemmer inside tokenizer. See examples section.

Details

S3 methods for creating an itoken iterator from list of tokens

- `list`: all elements of the input list should be character vectors containing tokens
- `character`: raw text source: the user must provide a tokenizer function
- `ifiles`: from files, a user must provide a function to read in the file (to `ifiles`) and a function to tokenize it (to `itoken`)
- `idir`: from a directory, the user must provide a function to read in the files (to `idir`) and a function to tokenize it (to `itoken`)
- `ifiles_parallel`: from files in parallel

See Also

[ifiles](#), [idir](#), [create_vocabulary](#), [create_dtm](#), [vectorizers](#), [create_tcm](#)

Examples

```
data("movie_review")
txt = movie_review$review[1:100]
ids = movie_review$id[1:100]
it = itoken(txt, tolower, word_tokenizer, n_chunks = 10)
it = itoken(txt, tolower, word_tokenizer, n_chunks = 10, ids = ids)
# Example of stemming tokenizer
# stem_tokenizer =function(x) {
#   word_tokenizer(x) %>% lapply(SnowballC::wordStem, language="en")
# }
#-----
# PARALLEL iterators
#-----
library(text2vec)

N_WORKERS = 1 # change 1 to number of cores in parallel backend
if(require(doParallel)) registerDoParallel(N_WORKERS)
data("movie_review")
it = itoken_parallel(movie_review$review[1:100], n_chunks = N_WORKERS)
system.time(dtm <- create_dtm(it, hash_vectorizer(2**16), type = 'dgTMatrix'))
```

LatentDirichletAllocation

Creates Latent Dirichlet Allocation model.

Description

Creates Latent Dirichlet Allocation model. At the moment only 'WarpLDA' is implemented. WarpLDA, an LDA sampler which achieves both the best $O(1)$ time complexity per token and the best $O(K)$ scope of random access. Our empirical results in a wide range of testing conditions demonstrate that WarpLDA is consistently 5-15x faster than the state-of-the-art Metropolis-Hastings based LightLDA, and is comparable or faster than the sparsity aware F+LDA.

Usage

```
LatentDirichletAllocation
LDA
LatentDirichletAllocationDistributed
```

Format

R6Class object.

Fields

`topic_word_distribution` distribution of words for each topic. Available after model fitting with `model$fit_transform()` method.

`components` unnormalized word counts for each topic-word entry. Available after model fitting with `model$fit_transform()` method.

Usage

For usage details see **Methods, Arguments and Examples** sections.

```
lda = LDA$new(n_topics = 10L, doc_topic_prior = 50 / n_topics, topic_word_prior = 1 / n_topics)
lda$fit_transform(x, n_iter = 1000, convergence_tol = 1e-3, n_check_convergence = 10, progressbar = interactive())
lda$transform(x, n_iter = 1000, convergence_tol = 1e-3, n_check_convergence = 5, progressbar = FALSE)
lda$get_top_words(n = 10, topic_number = 1L:private$n_topics, lambda = 1)
```

Methods

`$new(n_topics, doc_topic_prior = 50 / n_topics, # alpha topic_word_prior = 1 / n_topics, # beta met`
 Constructor for LDA model. For description of arguments see **Arguments** section.

`$fit_transform(x, n_iter, convergence_tol = -1, n_check_convergence = 0, progressbar = interactive()`
 fit LDA model to input matrix `x` and transforms input documents to topic space. Result is a matrix where each row represents corresponding document. Values in a row form distribution over topics.

`$transform(x, n_iter, convergence_tol = -1, n_check_convergence = 0, progressbar = FALSE)`
 transforms new documents into topic space. Result is a matrix where each row is a distribution of a documents over latent topic space.

`$get_top_words(n = 10, topic_number = 1L:private$n_topics, lambda = 1)` returns "top words" for a given topic (or several topics). Words for each topic can be sorted by probability of chance to observe word in a given topic (`lambda = 1`) and by "relevance" which also takes into account frequency of word in corpus (`lambda < 1`). From our experience in most cases setting `0.2 < lambda < 0.4` works well. See <http://nlp.stanford.edu/events/illvi2014/papers/sievert-illvi2014.pdf> for details.

`$plot(lambda.step = 0.1, reorder.topics = FALSE, ...)` plot LDA model using <https://cran.r-project.org/package=LDAvis> package. ... will be passed to `LDAvis::createJSON` and `LDAvis::serVis` functions

Arguments

lda A LDA object

x An input document-term matrix (should have column names = terms). **CSR** RsparseMatrix **used internally**, other formats will be tried to convert to CSR via as() function call.

n_topics integer desired number of latent topics. Also knows as **K**

doc_topic_prior numeric prior for document-topic multinomial distribution. Also knows as **alpha**

topic_word_prior numeric prior for topic-word multinomial distribution. Also knows as **eta**

n_iter integer number of sampling iterations

n_check_convergence defines how often calculate score to check convergence

convergence_tol numeric = -1 defines early stopping strategy. We stop fitting when one of two following conditions will be satisfied: (a) we have used all iterations, or (b) $\text{score_previous_check} / \text{score_current}$

Examples

```
library(text2vec)
data("movie_review")
N = 500
tokens = movie_review$review[1:N] %>% tolower %>% word_tokenizer
it = itoken(tokens, ids = movie_review$id[1:N])
v = create_vocabulary(it) %>%
  prune_vocabulary(term_count_min = 5, doc_proportion_max = 0.2)
dtm = create_dtm(it, vocab_vectorizer(v))
lda_model = LDA$new(n_topics = 10)
doc_topic_distr = lda_model$fit_transform(dtm, n_iter = 20)
# run LDAvis visualisation if needed (make sure LDAvis package installed)
# lda_model$plot()
```

LatentSemanticAnalysis

Latent Semantic Analysis model

Description

Creates LSA(Latent semantic analysis) model. See https://en.wikipedia.org/wiki/Latent_semantic_analysis for details.

Usage

```
LatentSemanticAnalysis
```

```
LSA
```

Format

R6Class object.

Usage

For usage details see **Methods, Arguments and Examples** sections.

```
lsa = LatentSemanticAnalysis$new(n_topics, method = c("randomized", "irlba"))
lsa$fit_transform(x, ...)
lsa$transform(x, ...)
lsa$components
```

Methods

```
$new(n_topics) create LSA model with n_topics latent topics
$fit_transform(x, ...) fit model to an input sparse matrix (preferably in dgCMatix format)
and then transform x to latent space
$transform(x, ...) transform new data x to latent space
```

Arguments

lsa A LSA object.

x An input document-term matrix. Preferably in dgCMatix format

n_topics integer desired number of latent topics.

method character, one of c("randomized", "irlba"). Defines underlying SVD algorithm. For very large data "randomized" usually works faster and more accurate.

... Arguments to internal functions. Notably useful for fit_transform() - these arguments will be passed to **irlba** or **svdr** functions which are used as backend for SVD.

Examples

```
data("movie_review")
N = 100
tokens = movie_review$review[1:N] %>% tolower %>% word_tokenizer
dtm = create_dtm(itoken(tokens), hash_vectorizer())
n_topics = 10
lsa_1 = LatentSemanticAnalysis$new(n_topics)
d1 = lsa_1$fit_transform(dtm)
# the same, but wrapped with S3 methods
d2 = fit_transform(dtm, lsa_1)
```

movie_review

IMDB movie reviews

Description

The labeled dataset consists of 5000 IMDB movie reviews, specially selected for sentiment analysis. The sentiment of the reviews is binary, meaning an IMDB rating < 5 results in a sentiment score of 0, and a rating >=7 has a sentiment score of 1. No individual movie has more than 30 reviews. Important note: we removed non ASCII symbols from the original dataset to satisfy CRAN policy.

Usage

```
data("movie_review")
```

Format

A data frame with 5000 rows and 3 variables:

id Unique ID of each review

sentiment Sentiment of the review; 1 for positive reviews and 0 for negative reviews

review Text of the review (UTF-8)

Source

<http://ai.stanford.edu/~amaas/data/sentiment/>

normalize

Matrix normalization

Description

normalize matrix rows using given norm

Usage

```
normalize(m, norm = c("l1", "l2", "none"))
```

Arguments

m matrix (sparse or dense).

norm character the method used to normalize term vectors

Value

normalized matrix

See Also

[create_dtm](#)

perplexity

Perplexity of a topic model

Description

Given document-term matrix, topic-word distribution, document-topic distribution calculates perplexity

Usage

```
perplexity(X, topic_word_distribution, doc_topic_distribution)
```

Arguments

X sparse document-term matrix which contains terms counts. Internally `Matrix::RsparseMatrix` is used. If `class(X) != 'RsparseMatrix'` function will try to coerce `X` to `RsparseMatrix` via `as()` call.

topic_word_distribution dense matrix for topic-word distribution. Number of rows = `n_topics`, number of columns = `vocabulary_size`. Sum of elements in each row should be equal to 1 - each row is a distribution of words over topic.

doc_topic_distribution dense matrix for document-topic distribution. Number of rows = `n_documents`, number of columns = `n_topics`. Sum of elements in each row should be equal to 1 - each row is a distribution of topics over document.

Examples

```
library(text2vec)
data("movie_review")
n_iter = 10
train_ind = 1:200
ids = movie_review$id[train_ind]
txt = movie_review[['review']][train_ind] %>% tolower
names(txt) = ids
tokens = word_tokenizer(txt)
it = itoken(tokens, progressbar = FALSE, ids = ids)
vocab = create_vocabulary(it) %>%
  prune_vocabulary(term_count_min = 5, doc_proportion_min = 0.02)
dtm = create_dtm(it, vectorizer = vocab_vectorizer(vocab))
n_topic = 10
model = LDA$new(n_topic, doc_topic_prior = 0.1, topic_word_prior = 0.01)
doc_topic_distr =
  model$fit_transform(dtm, n_iter = n_iter, n_check_convergence = 1,
                     convergence_tol = -1, progressbar = FALSE)
topic_word_distr_10 = model$topic_word_distribution
perplexity(dtm, topic_word_distr_10, doc_topic_distr)
```

`prepare_analogy_questions`*Prepares list of analogy questions*

Description

This function prepares a list of questions from a questions-words.txt format. For full examples see [GloVe](#).

Usage

```
prepare_analogy_questions(questions_file_path, vocab_terms)
```

Arguments

`questions_file_path`

character path to questions file.

`vocab_terms`

character words which we have in the vocabulary and word embeddings matrix.

See Also

[check_analogy_accuracy](#), [GloVe](#)

`prune_vocabulary`*Prune vocabulary*

Description

This function filters the input vocabulary and throws out very frequent and very infrequent terms. See examples in for the [vocabulary](#) function. The parameter `vocab_term_max` can also be used to limit the absolute size of the vocabulary to only the most frequently used terms.

Usage

```
prune_vocabulary(vocabulary, term_count_min = 1L, term_count_max = Inf,  
doc_proportion_min = 0, doc_proportion_max = 1, doc_count_min = 1L,  
doc_count_max = Inf, vocab_term_max = Inf)
```

Arguments

vocabulary	a vocabulary from the vocabulary function.
term_count_min	minimum number of occurrences over all documents.
term_count_max	maximum number of occurrences over all documents.
doc_proportion_min	minimum proportion of documents which should contain term.
doc_proportion_max	maximum proportion of documents which should contain term.
doc_count_min	term will be kept number of documents contain this term is larger than this value
doc_count_max	term will be kept number of documents contain this term is smaller than this value
vocab_term_max	maximum number of terms in vocabulary.

See Also

[vocabulary](#)

RelaxedWordMoversDistance

Creates model which can be used for calculation of "relaxed word movers distance".

Description

Relaxed word movers distance tries to measure distance between documents by calculating how hard is to transform words from first document into words from second document and vice versa. For more detail see original article: <http://mkusner.github.io/publications/WMD.pdf>.

Usage

```
RelaxedWordMoversDistance
```

```
RWMD
```

Format

[R6Class](#) object.

Fields

progressbar logical = TRUE whether to display progressbar

Usage

For usage details see **Methods, Arguments and Examples** sections.

```
rwmd = RelaxedWordMoversDistance$new(wv, method = c("cosine", "euclidean"), normalize = TRUE, progress = FALSE)
rwmd$dist2(x, y)
rwmd$pdist2(x, y)
```

Methods

`$new(wv, method = c("cosine", "euclidean"))` Constructor for RWMD model For description of arguments see **Arguments** section

`$dist2(x, y)` Computes distance between each row of sparse matrix x and each row of sparse matrix y

`$pdist2(x, y)` Computes "parallel" distance between rows of sparse matrix x and corresponding rows of the sparse matrix y

Arguments

rwmd RWMD object

x x sparse document term matrix

y y = NULL sparse document term matrix. If y = NULL (as by default), we will assume y = x

wv word vectors. Numeric matrix which contains word embeddings. Rows - words, columns - corresponding vectors. Rows should have word names.

method name of the distance for measuring similarity between two word vectors. In original paper authors use "euclidean", however we use "cosine" by default (better from our experience). This means $distance = 1 - cosine_angle_between_wv$

Examples

```
## Not run:
data("movie_review")
tokens = movie_review$review %>%
  tolower %>%
  word_tokenizer
v = create_vocabulary(itoken(tokens)) %>%
  prune_vocabulary(term_count_min = 5, doc_proportion_max = 0.5)
it = itoken(tokens)
vectorizer = vocab_vectorizer(v)
dtm = create_dtm(it, vectorizer)
tcm = create_tcm(it, vectorizer, skip_grams_window = 5)
glove_model = GloVe$new(word_vectors_size = 50, vocabulary = v, x_max = 10)
wv = glove_model$fit_transform(tcm, n_iter = 10)
# get average of main and context vectors as proposed in GloVe paper
wv = wv + t(glove_model$components)
rwmd_model = RWMD$new(wv)
rwmd_dist = dist2(dtm[1:100, ], dtm[1:10, ], method = rwmd_model, norm = 'none')
head(rwmd_dist)

## End(Not run)
```

 similarities

Pairwise Similarity Matrix Computation

Description

sim2 calculates pairwise similarities between the rows of two data matrices. **Note** that some methods work only on sparse matrices and others work only on dense matrices.

psim2 calculates "parallel" similarities between the rows of two data matrices.

Usage

```
sim2(x, y = NULL, method = c("cosine", "jaccard"), norm = c("l2", "none"))
```

```
psim2(x, y, method = c("cosine", "jaccard"), norm = c("l2", "none"))
```

Arguments

x	first matrix.
y	second matrix. For sim2 y = NULL set by default. This means that we will assume y = x and calculate similarities between all rows of the x.
method	character, the similarity measure to be used. One of c("cosine", "jaccard").
norm	character = c("l2", "none") - how to scale input matrices. If they already scaled - use "none"

Details

Computes the similarity matrix using given method.

psim2 takes two matrices and return a single vector. giving the 'parallel' similarities of the vectors.

Value

sim2 returns matrix of similarities between each row of matrix x and each row of matrix y.

psim2 returns vector of "parallel" similarities between rows of x and y.

 split_into

Split a vector for parallel processing

Description

This function splits a vector into n parts of roughly equal size. These splits can be used for parallel processing. In general, n should be equal to the number of jobs you want to run, which should be the number of cores you want to use.

Usage

```
split_into(vec, n)
```

Arguments

vec	input vector
n	integer desired number of chunks

Value

list with n elements, each of roughly equal length

text2vec	<i>text2vec</i>
----------	-----------------

Description

Fast vectorization, topic modeling, distances and GloVe word embeddings in R.

Details

To learn more about text2vec visit project website: text2vec.org Or start with the vignettes: `browseVignettes(package = "text2vec")`

TfIdf	<i>TfIdf</i>
-------	--------------

Description

Creates TfIdf(Latent semantic analysis) model. The IDF is defined as follows: $idf = \log(\# \text{ documents in the corpus}) / (\# \text{ documents where the term appears} + 1)$

Usage

```
TfIdf
```

Format

[R6Class](#) object.

Details

Term Frequency Inverse Document Frequency

Usage

For usage details see **Methods, Arguments and Examples** sections.

```
tfidf = TfIdf$new(smooth_idf = TRUE, norm = c('l1', 'l2', 'none'), sublinear_tf = FALSE)
tfidf$fit_transform(x)
tfidf$transform(x)
```

Methods

`$new(smooth_idf = TRUE, norm = c("l1", "l2", "none"), sublinear_tf = FALSE)` Creates tf-idf model

`$fit_transform(x)` fit model to an input sparse matrix (preferably in "dgCMatrix" format) and then transforms it.

`$transform(x)` transform new data x using tf-idf from train data

Arguments

tfidf A TfIdf object

x An input term-co-occurrence matrix. Preferably in dgCMatrix format

smooth_idf TRUE smooth IDF weights by adding one to document frequencies, as if an extra document was seen containing every term in the collection exactly once. This prevents division by zero.

norm c("l1", "l2", "none") Type of normalization to apply to term vectors. "l1" by default, i.e., scale by the number of words in the document.

sublinear_tf FALSE Apply sublinear term-frequency scaling, i.e., replace the term frequency with $1 + \log(\text{TF})$

Examples

```
data("movie_review")
N = 100
tokens = movie_review$review[1:N] %>% tolower %>% word_tokenizer
dtm = create_dtm(itoken(tokens), hash_vectorizer())
model_tfidf = TfIdf$new()
dtm_tfidf = model_tfidf$fit_transform(dtm)
```

tokenizers

Simple tokenization functions for string splitting

Description

very thin wrappers around base regular expressions. **For much more faster and functional tokenizers see tokenizers package: <https://cran.r-project.org/package=tokenizers>.** The reason for not including this to text2vec is to keep number of dependencies small. Also check `stringi::stri_split_*` and `stringr::str_split_*`.

Usage

```
word_tokenizer(strings, ...)

regexp_tokenizer(strings, pattern, ...)

char_tokenizer(strings, ...)

space_tokenizer(strings, sep = " ", xptr = FALSE, ...)
```

Arguments

strings	character vector
...	other parameters to strsplit function, which is used under the hood.
pattern	character pattern symbol.
sep	character, nchar(sep) = 1 - split strings by this character.
xptr	logical tokenize at C++ level - could speed-up by 15-50%.

Value

list of character vectors. Each element of list contains vector of tokens.

Examples

```
doc = c("first second", "bla, bla, blaa")
# split by words
word_tokenizer(doc)
#faster, but far less general - perform split by a fixed single whitespace symbol.
space_tokenizer(doc, " ")
```

transform

Transforms Matrix-like object using model

Description

Transforms Matrix-like object using model

Usage

```
## S3 method for class 'Matrix'
transform(`_data`, model, ...)

## S3 method for class 'matrix'
transform(`_data`, model, ...)
```

Arguments

<code>_data</code>	= <code>x</code> in other methods. A matrix like object, should inherit from <code>Matrix</code> or <code>matrix</code>
<code>model</code>	object of class <code>transformer</code> which implements method <code>\$transform(x, ...)</code>
<code>...</code>	additional data/model dependent arguments to downstream functions.

vectorizers

Vocabulary and hash vectorizers

Description

This function creates an object (closure) which defines on how to transform list of tokens into vector space - i.e. how to map words to indices. It supposed to be used only as argument to [create_dtm](#), [create_tcm](#), [create_vocabulary](#).

Usage

```
vocab_vectorizer(vocabulary)
```

```
hash_vectorizer(hash_size = 2^18, ngram = c(1L, 1L), signed_hash = FALSE)
```

Arguments

<code>vocabulary</code>	<code>text2vec_vocabulary</code> object, see create_vocabulary .
<code>hash_size</code>	integer The number of of hash-buckets for the feature hashing trick. The number must be greater than 0, and preferably it will be a power of 2.
<code>ngram</code>	integer vector. The lower and upper boundary of the range of n-values for different n-grams to be extracted. All values of n such that <code>ngram_min <= n <= ngram_max</code> will be used.
<code>signed_hash</code>	logical, indicating whether to use a signed hash-function to reduce collisions when hashing.

Value

A vectorizer object (closure).

See Also

[create_dtm](#) [create_tcm](#) [create_vocabulary](#)

Examples

```
data("movie_review")
N = 100
vectorizer = hash_vectorizer(2 ^ 18, c(1L, 2L))
it = itoken(movie_review$review[1:N], preprocess_function = tolower,
            tokenizer = word_tokenizer, n_chunks = 10)
hash_dtm = create_dtm(it, vectorizer)

it = itoken(movie_review$review[1:N], preprocess_function = tolower,
            tokenizer = word_tokenizer, n_chunks = 10)
v = create_vocabulary(it, c(1L, 1L) )

vectorizer = vocab_vectorizer(v)

it = itoken(movie_review$review[1:N], preprocess_function = tolower,
            tokenizer = word_tokenizer, n_chunks = 10)

dtm = create_dtm(it, vectorizer)
```

Index

*Topic **datasets**

- Collocations, 4
 - GlobalVectors, 13
 - LatentDirichletAllocation, 19
 - LatentSemanticAnalysis, 21
 - movie_review, 22
 - RelaxedWordMoversDistance, 26
 - TfIdf, 29
- as.lda_c, 2
- char_tokenizer (tokenizers), 30
- check_analogy_accuracy, 3, 25
- collect, 3
- Collocations, 4
- create_dtm, 6, 8, 17, 19, 23, 32
- create_tcm, 7, 8, 17, 19, 32
- create_vocabulary, 9, 17, 19, 32
- dist, 11
- dist2 (distances), 11
- distances, 11
- fit, 12
- fit_transform, 12
- foreach, 6, 8, 10
- GlobalVectors, 13
- GloVe, 7, 25
- GloVe (GlobalVectors), 13
- glove, 3, 15
- hash_vectorizer (vectorizers), 32
- idir, 19
- idir (ifiles), 16
- ifiles, 16, 19
- ifiles_parallel (ifiles), 16
- irlba, 22
- itoken, 6, 8, 9, 16, 17, 17, 19
- itoken_parallel (itoken), 17
- LatentDirichletAllocation, 19
- LatentDirichletAllocationDistributed (LatentDirichletAllocation), 19
- LatentSemanticAnalysis, 21
- LDA (LatentDirichletAllocation), 19
- LSA (LatentSemanticAnalysis), 21
- movie_review, 22
- normalize, 23
- pdist2 (distances), 11
- perplexity, 24
- prepare_analogy_questions, 3, 25
- prune_vocabulary, 25
- psim2 (similarities), 28
- R6Class, 4, 13, 20, 21, 26, 29
- regex_tokenizer (tokenizers), 30
- RelaxedWordMoversDistance, 26
- RWMD, 11
- RWMD (RelaxedWordMoversDistance), 26
- sim2 (similarities), 28
- similarities, 28
- space_tokenizer (tokenizers), 30
- split_into, 28
- strsplit, 31
- svdr, 22
- text2vec, 29
- text2vec-package (text2vec), 29
- TfIdf, 29
- tokenizers, 30
- transform, 31
- vectorizers, 6, 8, 17, 19, 32
- vocab_vectorizer (vectorizers), 32
- vocabulary, 25, 26
- vocabulary (create_vocabulary), 9
- word_tokenizer (tokenizers), 30