Package 'text2vec'

October 4, 2016

Type Package

Version 0.4.0

Date 2016-10-04

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 (>= 2.1.2), R6 (>= 2.1.2)

LinkingTo Rcpp, RcppParallel, digest

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

URL http://text2vec.org

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

VignetteBuilder knitr

LazyData true

RoxygenNote 5.0.1

NeedsCompilation yes

Author Dmitriy Selivanov [aut, cre],
Lincoln Mullen [ctb]

Repository CRAN

Date/Publication 2016-10-04 17:48:07

R topics documented:

`as.lda_c` *Converts document-term matrix sparse matrix to 'lda_c' format*

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, verbose = TRUE)
```

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.
`verbose`	`logical` whether to print messages during evaluation.

See Also

prepare_analogy_questions, glove

`create_corpus` *Create a corpus*

Description

This functions creates corpus objects (based on vocabulary or hashes), which are stored outside of R's heap and wrapped via reference classes using Rcpp-Modules. From those objects you can easily extract document-term (DTM) and term-co-occurrence (TCM) matrices. Also, text2vec grows the corpus for DTM and TCM matrices simultaneously in a RAM-friendly and efficient way using the iterators abstraction. You can build corpora from objects or files which are orders of magnitude larger that available RAM.

Usage

```
create_corpus(iterator, vectorizer)
```

Arguments

iterator	iterator over a list of character vectors. Each element is a list of tokens, that is, tokenized and normalized strings.
vectorizer	function vectorizer function. See vectorizers.

Value

Corpus object.

See Also

vectorizers, create_dtm, get_dtm, get_tcm, create_tcm

---

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", "lda_c"), ...)

## S3 method for class 'itoken'
create_dtm(it, vectorizer, type = c("dgCMatrix", "dgTMatrix",
  "lda_c"), ...)

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

Arguments

it	itoken iterator or list of itoken iterators.
vectorizer	function vectorizer function; see vectorizers.
type	character, one of c("dgCMatrix", "dgTMatrix","lda_c"). "lda_c" is Blei's lda-c format (a list of 2 * doc_terms_size); see https://www.cs.princeton.edu/~blei/lda-c/readme.txt
...	arguments to the foreach function which is used to iterate over it.
verbose	logical print status messages

Details

If a parallel backend is registered and first argument is a list of `itoken`, itereators, fucntion 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 create_corpus get_dtm

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
doParallel::registerDoParallel(N_WORKERS)
splits = split_into(movie_review$review, N_WORKERS)
jobs = lapply(splits, itoken, tolower, word_tokenizer, chunks_number = 1)
vectorizer = hash_vectorizer()
dtm = create_dtm(jobs, vectorizer, type = 'dgTMatrix')

## End(Not run)
```

---

create_tcm *Term-co-occurence 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, ...)

## S3 method for class 'itoken'
create_tcm(it, vectorizer, ...)

## S3 method for class 'list'
create_tcm(it, vectorizer, verbose = FALSE,
  work_dir = tempdir(), ...)
```

Arguments

it	list of iterators over tokens from itoken. Each element is a list of tokens, that is, tokenized and normalized strings.
vectorizer	function vectorizer function. See vectorizers.
...	arguments to foreach function which is used to iterate over it.
verbose	logical print status messages
work_dir	working directory for intermediate results

Details

If a parallel backend is registered, it will onstruct 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, grow_dtm = FALSE, skip_grams_window = 3L)
tcm = create_tcm(itoken(tokens), vectorizer)

# parallel version

# set to number of cores on your machine
```

```
N_WORKERS = 1
splits = split_into(movie_review$review, N_WORKERS)
jobs = lapply(splits, itoken, tolower, word_tokenizer)
v = create_vocabulary(jobs)
vectorizer = vocab_vectorizer(v, grow_dtm = FALSE, skip_grams_window = 3L)
jobs = lapply(splits, itoken, tolower, word_tokenizer)
doParallel::registerDoParallel(N_WORKERS)
tcm = create_tcm(jobs, vectorizer)

## 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 = "_", ...)
```

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 ngram_min <= n <= ngram_max 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 foreach function.

Value

text2vec_vocabulary object, which is actually a list with following fields:

1. vocab: a data.frame which contains columns
   - terms character vector of unique terms
   - terms_counts integer vector of term counts across all documents
   - doc_counts integer vector of document counts that contain corresponding term
2. ngram: integer vector, the lower and upper boundary of the range of n-gram-values.
3. document_count: integer number of documents vocabulary was built.

Methods (by class)

- character: creates text2vec_vocabulary from predefined character vector. Terms will be inserted **as is**, without any checks (ngrams numner, 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.

Examples

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

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

Arguments

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

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 *Fits model to data*

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 Matrix or matrix
model	instance of class estimator which should implement method with signature $fit(x, y, ...)
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

invisible(object$self())

---

fit_transform *Fit model to data, then transform it*

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

get_dtm *Extract document-term matrix*

Description

This function extracts a document-term matrix from a Corpus object.

Usage

```
get_dtm(corpus, type = c("dgCMatrix", "dgTMatrix", "lda_c"))
```

Arguments

corpus	HashCorpus or VocabCorpus object. See create_corpus for details.
type	character, one of c("dgCMatrix", "dgTMatrix", "lda_c"). "lda_c" is Blei's lda-c format (a list of 2 * doc_terms_size); see https://www.cs.princeton.edu/~blei/lda-c/readme.txt

Examples

```
N = 1000
tokens = movie_review$review[1:N] %>% tolower %>% word_tokenizer
it = itoken(tokens)
v = create_vocabulary(it)

#remove very common and uncommon words
pruned_vocab = prune_vocabulary(v, term_count_min = 10,
 doc_proportion_max = 0.8, doc_proportion_min = 0.001,
 max_number_of_terms = 10000)

vectorizer = vocab_vectorizer(v)
it = itoken(tokens)
corpus = create_corpus(it, vectorizer)
dtm = get_dtm(corpus)
```

get_idf *Inverse document-frequency scaling matrix*

Description

This function creates an inverse-document-frequency (IDF) scaling matrix from a document-term matrix. The IDF is defined as follows: idf = log(# documents in the corpus) / (# documents where the term appears + 1)

Usage

```
get_idf(dtm, log_scale = log, smooth_idf = TRUE)
```

Arguments

dtm	a document-term matrix of class dgCMatrix or dgTMatrix.
log_scale	function to use in calculating the IDF matrix. Usually log is used, but it might be worth trying log2.
smooth_idf	logical 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.

Value

ddiMatrix IDF scaling diagonal sparse matrix.

See Also

get_tf, get_dtm, create_dtm

---

get_tcm *Extract term-co-occurence matrix*

Description

This function creates a term-co-occurence matrix from a Corpus object.

Usage

```
get_tcm(corpus)
```

Arguments

corpus	HashCorpus or VocabCorpus object. See create_corpus, vectorizers for details.

See Also

create_corpus

Examples

```
## Not run:
txt = movie_review[['review']][1:1000]
it = itoken(txt, tolower, word_tokenizer)
vocab = create_vocabulary(it)
#remove very common and uncommon words
pruned_vocab = prune_vocabulary(vocab, term_count_min = 10, doc_proportion_max = 0.8,
                                doc_proportion_min = 0.001, max_number_of_terms = 5000)

vectorizer = vocab_vectorizer(pruned_vocab, grow_dtm = FALSE, skip_grams_window = 5L)
it = itoken(txt, tolower, word_tokenizer)
corpus = create_corpus(it, vectorizer)
```

```
tcm = get_tcm(corpus)
dim(tcm)

## End(Not run)
```

`get_tf` *Term-frequency scaling matrix*

Description

This function creates a term-frequency (TF) scaling matrix from a document-term matrix.

Usage

```
get_tf(dtm, norm = c("l1", "l2"))
```

Arguments

`dtm`	a document-term matrix of class `dgCMatrix` or `dgTMatrix`.
`norm`	`character` the method used to normalize term vectors. `"l1"` by default, i.e., scale by the number of words in the document.

Value

`ddiMatrix` TF scaling diagonal sparse matrix.

See Also

get_idf, get_dtm, create_dtm

`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()` method.

Usage

```
GloVe
```

Format

`R6Class` object.

Fields

`dump_every_n 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. Generelly 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 paramenter.**

`verbose logical = TRUE` whether to display training inforamtion

Usage

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

```
glove = GlobalVectors$new(word_vectors_size, vocabulary, x_max)
glove$fit(x, n_iter)
glove$get_word_vectors()
glove$dump_model()
```

Methods

`$new(word_vectors_size, vocabulary, x_max, learning_rate = 0.15, max_cost = 10, alpha = 0.75, lambd` Constructor for Global vectors model. For description of arguments see **Arguments** section.

`$fit(x, n_iter, convergence_tol = -1)` fit Glove model to input matrix `x`

`$get_word_vectors()` get word vector - obtain GloVe word embeddings

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

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

Arguments

glove A `GloVe` object

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

n_iter `integer` number of SGD iterations

word_vectors_size desired dimenson for word vectors

vocabulary `character` vector or instance of `text2vec_vocabulary` class. Each word should correspond to dimension of co-occurence 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_it` `convergence_tol`. By default perform all iterations.

max_cost `numeric = 10` the maximum absolute value of calculated gradient for any single co-occurrence pair. Try to set this to a smaller value if you have problems with numerical stability

alpha `numeric = 0.75` the alpha in weighting function formula : $f(x) = 1 if x > x_max; else (x/x_max)^alpha$

lambda `numeric = 0.0`, L1 regularization coefficient. `0` = vanilla GloVe, corrsesponds 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 that vanilla GloVe on small corpuses

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://mattmahoney.net/dc/text8.zip', temp)
text8 = readLines(unz(temp, "text8"))
it = itoken(text8)
vocab = create_vocabulary(it) %>%
 prune_vocabulary(term_count_min = 5)
v_vect = vocab_vectorizer(vocab, grow_dtm = FALSE, skip_grams_window = 5L)
tcm = create_tcm(it, v_vect)

glove_model = GloVe(word_vectors_size = 50, vocabulary = vocab, x_max = 10, learning_rate = .25)
# fit model and get word vectors
fit(tcm, glove_model, n_iter = 10)
wv = glove_model$get_word_vectors()

## End(Not run)
```

`glove` *Fit a GloVe word-embedded model*

Description

DEPRECIATED.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, verbose = TRUE,
  convergence_threshold = -1, grain_size = 100000L, max_cost = 10,
  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 dimenson 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)! Generelly 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.
verbose	logical whether to display training inforamtion
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) cost_previous_iter / cost_current_iter convergence_threshold.
grain_size	I do not recommend adjusting this paramenter. This is the grain_size for RcppParallel::parallelReduc For details, see http://rcppcore.github.io/RcppParallel/#grain-size.
max_cost	the maximum absolute value of calculated gradient for any single co-occurrence pair. Try to set this to a smaller value if you have problems with numerical stability.
alpha	the alpha in weighting function formula : $f(x) = 1 if x > x_max; else (x/x_max)^alpha$
...	arguments passed to other methods (not used at the moment).

---

ifiles *Creates iterator over text files from the disk*

Description

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

Usage

```
ifiles(file_paths, reader = readLines)

idir(path, reader = readLines)
```

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 provied names 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.

See Also

itoken

Examples

```
current_dir_files = list.files(path = ".", full.names = TRUE)
files_iterator = ifiles(current_dir_files)
dir_files_iterator = idir(path = ".")
```

---

itoken *Iterators over input objects*

Description

This function creates iterators over input objects to vocabularies, corpora, or DTM and TCM matrices. This iterator is usually used in following functions : create_vocabulary, create_corpus, create_dtm, vectorizers, create_tcm. See them for details.

Usage

```
itoken(iterable, ...)

## S3 method for class 'list'
itoken(iterable, chunks_number = 10,
  progressbar = interactive(), ids = NULL, ...)

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

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

Arguments

`iterable`	an object from which to generate an iterator
`...`	arguments passed to other methods (not used at the moment)
`chunks_number`	`integer`, the number of pieces that object should be divided into.
`progressbar`	`logical` indicates whether to show progress bar.
`ids`	`vector` of document ids. If `ids` is not provided, `names(iterable)` will be used. If `names(iterable) == NULL`, incremental ids will be assigned.
`preprocessor`	`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.
`tokenizer`	`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)

See Also

ifiles, idir, create_vocabulary, create_corpus, 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, chunks_number = 10)
it = itoken(txt, tolower, word_tokenizer, chunks_number = 10, ids = ids)
# Example of stemming tokenizer
# stem_tokenizer = function(x) {
#  word_tokenizer(x) %>% lapply(SnowballC::wordStem('en'))
# }
```

`LatentDirichletAllocation`

Creates Latent Dirichlet Allocation model.

Description

Creates Latent Dirichlet Allocation model.

Usage

```
LatentDirichletAllocation

LDA
```

Format

`R6Class` object.

Fields

`verbose logical = TRUE` whether to display training inforamtion

Usage

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

```
lda = LatentDirichletAllocation$new(n_topics, vocabulary,
              doc_topic_prior = 1 / n_topics, topic_word_prior = 1 / n_topics)
lda$fit(x, n_iter, convergence_tol = -1, check_convergence_every_n = 0)
lda$fit_transform(x, n_iter, convergence_tol = -1, check_convergence_every_n = 0)
lda$get_word_vectors()
```

Methods

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

`$fit(x, n_iter, convergence_tol = -1, check_convergence_every_n = 0)` fit LDA model to input matrix `x`

`$fit_transform(x, n_iter, convergence_tol = -1, check_convergence_every_n = 0)` fit LDA model to input matrix `x` and transforms input documents to topic space

`$transform(x, n_iter = 100, convergence_tol = 0.005, check_convergence_every_n = 1)`
transforms new documents to topic space

`$get_word_vectors()` get word-topic distribution

`$plot(...)` 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.

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

vocabulary vocabulary in a form of character or text2vec_vocab

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 Gibbs iterations

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) perplexity_previous_iter / perplexit convergence_tol. By default perform all iterations.

check_convergence_every_n integer Defines frequency of perplexity calculation. In some cases perplexity calculation during LDA fitting can take noticable amount of time. It make sense to do not calculate it at each iteration.

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_c')
lda_model = LatentDirichletAllocation$new(n_topics = 10, vocabulary = v,
 doc_topic_prior = 0.1,
 topic_word_prior = 0.1)
 doc_topic_distr = lda_model$fit_transform(dtm, n_iter =20, check_convergence_every_n = 5)
 # 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.

Fields

`verbose logical = TRUE` whether to display training inforamtion

Usage

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

```
lsa = LatentSemanticAnalysis$new(n_topics)
lsa$fit_transform(x)
lsa$get_word_vectors()
```

Methods

`$new(n_topics)` create LSA model with `n_topics` latent topics

`$fit(x, ...)` fit model to an input DTM (preferably in "dgCMatrix" format)

`$fit_transform(x, ...)` fit model to an input sparse matrix (preferably in "dgCMatrix" 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.

n_topics `integer` desired number of latent topics.

... Arguments to internal functions. Notably useful for `fit()`, `fit_transform()` - these arguments will be passed to irlba function which is 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)
fit(dtm, lsa_1) # or lsa_1$fit(dtm)
d1 = lsa_1$transform(dtm)
lsa_2 = LatentSemanticAnalysis$new(n_topics)
d2 = lsa_2$fit_transform(dtm)
all.equal(d1, d2)
# the same, but wrapped with S3 methods
all.equal(fit_transform(dtm, lsa_2), 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

get_idf, get_dtm, create_dtm

`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, verbose = TRUE)
```

Arguments

`questions_file_path`	`character` path to questions file.
`vocab_terms`	`character` words which we have in the vocabulary and word embeddings matrix.
`verbose`	`logical` whether to print messages during evaluation.

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 `max_number_of_terms` 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,
  max_number_of_terms = Inf)
```

Arguments

`vocabulary`	a vocabulary from the vocabulary function.
`term_count_min`	minimum number of occurences over all documents.
`term_count_max`	maximum number of occurences 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.
`max_number_of_terms`	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 transofrm 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

`verbose logical = TRUE` whether to display additional inforamtion during calculations.

Usage

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

```
rwmd = RelaxedWordMoversDistance$new(wv, method = c("cosine", "euclidean"))
rwmd$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_betwen_wv`

Examples

```
## Not run:
data("movie_review")
tokens = movie_review$review %>%
  tolower %>%
  word_tokenizer
v = create_vocabulary(itoken(tokens)) %>%
  prune_vocabulary(term_count_min = 5, doc_proportion_max = 0.5)
corpus = create_corpus(itoken(tokens), vocab_vectorizer(v, skip_grams_window = 5))
dtm = get_dtm(corpus)
tcm = get_tcm(corpus)
glove_model = GloVe$new(word_vectors_size = 50, vocabulary = v, x_max = 10)
wv = glove_model$fit(tcm, n_iter = 10)
rwmd_model = RWMD(wv)
rwmd_dist = dist2(dtm[1:10, ], dtm[1:100, ], method = rwmd_model, norm = 'none')

## 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"),
  verbose = TRUE)

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

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"
verbose	logical whether to display additional information during calculations

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` *text2vec*

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` *TfIdf*

Description

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

Usage

```
TfIdf
```

Format

`R6Class` object.

Details

Term Frequency Inverse Document Frequency

Fields

`verbose logical = TRUE` whether to display training inforamtion

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(x)
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(x)` fit tf-idf model to an input DTM (preferably in "dgCMatrix" format)

`$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-cooccurence 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(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()
model_tfidf$fit(dtm)
dtm_1 = model_tfidf$transform(dtm)
dtm_2 = model_tfidf$fit_transform(dtm)
identical(dtm_1, dtm_2)
```

---

tokenizers *Simple tokenization functions, which performs string splitting*

Description

simple wrappers around base regular expressions. For much more faster and functional tokenizers see `tokenizers` package: https://cran.r-project.org/package=tokenizers. Also see `str_split_*` functions in `stringi` and `stringr` packages. The reason for not including this packages to `text2vec` dependencies is our desare to keep number of dependencies as small as possible.

Usage

```
word_tokenizer(strings, ...)

regexp_tokenizer(strings, pattern, ...)

char_tokenizer(strings, ...)

space_tokenizer(strings, ...)
```

Arguments

strings	character vector
...	other parameters to strsplit function, which is used under the hood.
pattern	character pattern symbol.

Value

list of character vectors. Each element of list containts 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.
regexp_tokenizer(doc, " ", TRUE)
```

---

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

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

transform_filter_commons *Remove terms from a document-term matrix*

Description

This function removes very common and very uncommon words from a document-term matrix.

Usage

```
transform_filter_commons(dtm, term_freq = c(uncommon = 0.001, common = 0.975))
```

Arguments

dtm	a document-term matrix of class dgCMatrix or dgTMatrix.
term_freq	numeric vector of 2 values in between 0 and 1. The first element corresponds to frequency of uncommon words; the second element corresponds to the frequency of common words. Terms which are observed less than first value or frequency or more than second will be filtered out.

See Also

prune_vocabulary, transform_tf, transform_tfidf, transform_binary

transform_tf *Scale a document-term matrix*

Description

This set of functions scales a document-term matrix.

transform_tf: scale a DTM by one of two methods. If norm = "l1", then then dtm_tf = (count of a particular word / (total number of words in the document). If norm = "l2", then dtm_tf = (count of a particular word in the number words in the document) ^ 2.

transform_binary: scale a DTM so that if a cell is 1 if a word appears in the document; otherwise it is 0.

transform_tfidf: scale a DTM so that dtm_idf = log(count of a particular word in a document) / (number of d + 1)

Usage

```
transform_tf(dtm, sublinear_tf = FALSE, norm = c("l1", "l2", "none"))

transform_tfidf(dtm, idf = NULL, sublinear_tf = FALSE, norm = c("l1",
  "l2"))

transform_binary(dtm)
```

Arguments

dtm	a document-term matrix of class dgCMatrix or dgTMatrix.
sublinear_tf	logical, FALSE by default. Apply sublinear term-frequency scaling, i.e., replace the term frequency with 1 + log(TF).
norm	character Type of normalization to apply to term vectors. "l1" by default, i.e., scale by the number of words in the document.
idf	ddiMatrix a diagonal matrix for IDF scaling. See get_idf. If not provided the IDF scaling matrix will be calculated from the matrix passed to dtm.

Functions

- transform_tfidf: Scale a document-term matrix via TF-IDF
- transform_binary: Transform a document-term matrix into binary representation

See Also

get_idf, get_tf

Examples

```
## Not run:
data(moview_review)

txt = movie_review[["review"]][1:1000]
it = itoken(txt, tolower, word_tokenizer)
vocab = vocabulary(it)
#remove very common and uncommon words
pruned_vocab = prune_vocabulary(vocab,
 term_count_min = 10,
 doc_proportion_max = 0.8, doc_proportion_min = 0.001,
 max_number_of_terms = 20000)

it = itoken(txt, tolower, word_tokenizer)
dtm = create_dtm(it, pruned_vocab)

dtm_filtered = dtm %>%
 # functionality overlaps with prune_vocabulary(),
 # but still can be useful in some cases
 # filter out very common and very uncommon terms
 transform_filter_commons( c(0.001, 0.975) )

# simple term-frequency transormation
transformed_tf = dtm %>%
 transform_tf

# tf-idf transormation
idf = get_idf(dtm)
transformed_tfidf = transform_tfidf(dtm,  idf)

## End(Not run)
```

vectorizers *Vocabulary and hash vectorizers*

Description

This function creates a text vectorizer function which is used in constructing a dtm/tcm/corpus.

Usage

```
vocab_vectorizer(vocabulary, grow_dtm = TRUE, skip_grams_window = 0L)

hash_vectorizer(hash_size = 2^18, ngram = c(1L, 1L), signed_hash = FALSE,
  grow_dtm = TRUE, skip_grams_window = 0L)
```

Arguments

vocabulary	text2vec_vocabulary object, see create_vocabulary.
grow_dtm	logical Should we grow the document-term matrix during corpus construction or not.
skip_grams_window	integer window for term-co-occurence matrix construction. skip_grams_window should be > 0 if you plan to use vectorizer in create_tcm function. Value of 0L means to not construct the TCM.
hash_size	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.
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 ngram_min <= n <= ngram_max will be used.
signed_hash	logical, indicating whether to use a signed hash-function to reduce collisions when hashing.

Value

A vectorizer function

See Also

create_dtm create_tcm create_vocabulary create_corpus

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, chunks_number = 10)
corpus = create_corpus(it, vectorizer)
```

```
hash_dtm = get_dtm(corpus)

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

corpus = create_corpus(it, vectorizer)
voacb_dtm = get_dtm(corpus)
```

Index