Package 'text2vec'

February 18, 2020

Type Package

Version 0.6

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 C++11

Depends R (>= 3.6.0), methods

Imports Matrix (>= 1.1), Rcpp (>= 1.0.3), R6 (>= 2.3.0), data.table(>= 1.9.6), rsparse (>= 0.3.3.4), stringi (>= 1.1.5), mlapi (>= 0.1.0), lgr (>= 0.2), digest (>= 0.6.8)

LinkingTo Rcpp, digest (>= 0.6.8)

Suggests magrittr, udpipe (>= 0.6), glmnet, testthat, covr, knitr, rmarkdown, proxy

URL http://text2vec.org

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

VignetteBuilder knitr

LazyData true

RoxygenNote 6.1.1

NeedsCompilation yes

Author Dmitriy Selivanov [aut, cre, cph], Manuel Bickel [aut, cph] (Coherence measures for topic models), Qing Wang [aut, cph] (Author of the WaprLDA C++ code)

Repository CRAN

Date/Publication 2020-02-18 14:20:03 UTC

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

BNS *BNS*

Description

Creates BNS (bi-normal separation) model. Defined as: Q(true positive rate) - Q(false positive rate), where Q is a quantile function of normal distribution.

Usage

```
BNS
```

Format

`R6Class` object.

Details

Bi-Normal Separation

Fields

`bns_stat` `data.table` with computed BNS statistic. Useful for feature selection.

Usage

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

```
bns = BNS$new(treshold = 0.0005)
bns$fit_transform(x, y)
bns$transform(x)
```

Methods

`$new(treshold = 0.0005)` Creates bns model

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

`$transform(x)` transform new data `x` using bns from train data

Arguments

bns A `BNS` object

x An input document term matrix. Preferably in `dgCMatrix` format

y Binary target variable coercible to logical.

treshold Clipping treshold to avoid infinities in quantile function.

Examples

```
data("movie_review")
N = 1000
it = itoken(head(movie_review$review, N), preprocessor = tolower, tokenizer = word_tokenizer)
vocab = create_vocabulary(it)
dtm = create_dtm(it, vocab_vectorizer(vocab))
model_bns = BNS$new()
dtm_bns = model_bns$fit_transform(dtm, head(movie_review$sentiment, N))
```

---

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

---

coherence

Coherence metrics for topic models

Description

Given a topic model with topics represented as ordered term lists, the coherence may be used to assess the quality of individual topics. This function is an implementation of several of the numerous possible metrics for such kind of assessments. Coherence calculation is sensitive to the content of the reference tcm that is used for evaluation and that may be created with different parameter settings. Please refer to the details section (or reference section) for information on typical combinations of metric and type of tcm. For more general information on measuring coherence a starting point is given in the reference section.

Usage

```
coherence(x, tcm, metrics = c("mean_logratio", "mean_pmi", "mean_npmi",
  "mean_difference", "mean_npmi_cosim", "mean_npmi_cosim2"),
  smooth = 1e-12, n_doc_tcm = -1)
```

Arguments

x	A character matrix with the top terms per topic (each column represents one topic), e.g., as created by get_top_words(). Terms of x have to be ranked per topic starting with rank 1 in row 1.
tcm	The term co-occurrence matrix, e.g, a Matrix::sparseMatrix or base::matrix, serving as the reference to calculate coherence metrics. Please note that a memory efficient version of the tcm is assumed as input with all entries in the lower triangle (excluding diagonal) set to zero (see, e.g., create_tcm). Please also note that some efforts during any pre-processing steps might be skipped since the tcm is internally reduced to the top word space, i.e., all unique terms of x.
metrics	Character vector specifying the metrics to be calculated. Currently the following metrics are implemented: c("mean_logratio","mean_pmi","mean_npmi","mean_difference","mea Please refer to the details section for more information on the metrics.
smooth	Numeric smoothing constant to avoid logarithm of zero. By default, set to 1e-12.
n_doc_tcm	The integer number of documents or text windows that was used to create the tcm. n_doc_tcm is used to calculate term probabilities from term counts as required for several metrics.

Details

The currently implemented coherence metrics are described below including a description of the content type of the tcm that showed good performance in combination with a specific metric.
For details on how to create tcm see the example section.
For details on performance of metrics see the resources in the reference section that served for definition of standard settings for individual metrics.
Note that depending on the use case, still, different settings than the standard settings for creation of tcm may be reasonable.
Note that for all currently implemented metrics the tcm is reduced to the top word space on basis of the terms in x.

Considering the use case of finding the optimum number of topics among several models with different metrics, calculating the mean score over all topics and normalizing this mean coherence scores from different metrics might be considered for direct comparison.

Each metric usually opts for a different optimum number of topics. From initial experience it may be assumed that logratio, pmi and nmpi usually opt for smaller numbers, whereas the other metrics rather tend to propose higher numbers.

Implemented metrics:

- "mean_logratio"
  The logarithmic ratio is calculated as
  log(smooth + tcm[x,y]) -log(tcm[y,y]),

where x and y are term index pairs from a "preceding" term index combination.
Given the indices c(1,2,3), combinations are `list(c(2,1),c(3,1),c(3,2))`.

The `tcm` should represent the boolean term co-occurrence (internally the actual counts are used) in the original documents and, therefore, is an intrinsic metric in the standard use case.

This metric is similar to the UMass metric, however, with a smaller smoothing constant by default and using the mean for aggregation instead of the sum.

- "mean_pmi"
  The pointwise mutual information is calculated as
  `log2((tcm[x,y]/n_doc_tcm) + smooth) -log2(tcm[x,x]/n_doc_tcm) -log2(tcm[y,y]/n_doc_tcm)`,
  where x and y are term index pairs from an arbitrary term index combination
  that subsets the lower or upper triangle of `tcm`, e.g. "preceding".

  The `tcm` should represent term co-occurrences within a boolean sliding window of size `10` (internally probabilities are used) in an external reference corpus and, therefore, is an extrinsic metric in the standard use case.

  This metric is similar to the UCI metric, however, with a smaller smoothing constant by default and using the mean for aggregation instead of the sum.
- "mean_npmi"
  Similar (in terms of all parameter settings, etc.) to "mean_pmi" metric but using the normalized pmi instead, which is calculated as
  `(log2((tcm[x,y]/n_doc_tcm) + smooth) -log2(tcm[x,x]/n_doc_tcm) -log2(tcm[y,y]/n_doc_tcm)) / -log2((tcm[x,y]/n_doc_tcm) + smooth)`,

  This metric may perform better than the simpler pmi metric.
- "mean_difference"
  The difference is calculated as
  `tcm[x,y]/tcm[x,x] -(tcm[y,y]/n_tcm_windows)`,
  where x and y are term index pairs from a "preceding" term index combination.
  Given the indices c(1,2,3), combinations are `list(c(1,2),c(1,3),c(2,3))`.

  The `tcm` should represent the boolean term co-occurrence (internally probabilities are used) in the original documents and, therefore, is an intrinsic metric in the standard use case.
- "mean_npmi_cosim"
  First, the npmi of an individual top word with each of the top words is calculated as in "mean_npmi".
  This result in a vector of npmi values for each top word.
  On this basis, the cosine similarity between each pair of vectors is calculated.

  The `tcm` should represent term co-occurrences within a boolean sliding window of size `5` (internally probabilities are used) in an external reference corpus and, therefore, is an extrinsic metric in the standard use case.

- "mean_npmi_cosim2"
  First, a vector of npmi values for each top word is calculated as in "mean_npmi_cosim".

On this basis, the cosine similarity between each vector and the sum of all vectors is calculated (instead of the similarity between each pair).

The `tcm` should represent term co-occurrences within a boolean sliding window of size `110` (internally probabilities are used) in an external reference corpus and, therefore, is an extrinsic metric in the standard use case.

Value

A `numeric` `matrix` with the coherence scores of the specified `metrics` per topic.

References

Below mentioned paper is the main theoretical basis for this code.
Currently only a selection of metrics stated in this paper is included in this R implementation.
Authors: Roeder, Michael; Both, Andreas; Hinneburg, Alexander (2015)
Title: Exploring the Space of Topic Coherence Measures.
In: Xueqi Cheng, Hang Li, Evgeniy Gabrilovich und Jie Tang (Eds.):
Proceedings of the Eighth ACM International Conference on Web Search and Data Mining - WSDM '15.
the Eighth ACM International Conference. Shanghai, China, 02.02.2015 - 06.02.2015.
New York, USA: ACM Press, p. 399-408.
https://dl.acm.org/citation.cfm?id=2685324
This paper has been implemented by above listed authors as the Java program "palmetto".
See https://github.com/dice-group/Palmetto or http://aksw.org/Projects/Palmetto.html.

Examples

```
library(data.table)
library(text2vec)
library(Matrix)
data("movie_review")
N = 500
tokens = word_tokenizer(tolower(movie_review$review[1:N]))
it = itoken(tokens, progressbar = FALSE)
v = create_vocabulary(it)
v = prune_vocabulary(v, term_count_min = 5, doc_proportion_max = 0.2)
dtm = create_dtm(it, vocab_vectorizer(v))

n_topics = 10
lda_model = LDA$new(n_topics)
fitted = lda_model$fit_transform(dtm, n_iter = 20)
tw = lda_model$get_top_words(n = 10, lambda = 1)

# for demonstration purposes create intrinsic TCM from original documents
# scores might not make sense for metrics that are designed for extrinsic TCM
tcm = crossprod(sign(dtm))

# check coherence
```

```
logger = lgr::get_logger('text2vec')
logger$set_threshold('debug')
res = coherence(tw, tcm, n_doc_tcm = N)
res

# example how to create TCM for extrinsic measures from an external corpus
external_reference_corpus = tolower(movie_review$review[501:1000])
tokens_ext = word_tokenizer(external_reference_corpus)
iterator_ext = itoken(tokens_ext, progressbar = FALSE)
v_ext = create_vocabulary(iterator_ext)
# for reasons of efficiency vocabulary may be reduced to the terms matched in the original corpus
v_ext= v_ext[v_ext$term %in% v$term, ]
# external vocabulary may be pruned depending on the use case
v_ext = prune_vocabulary(v_ext, term_count_min = 5, doc_proportion_max = 0.2)
vectorizer_ext = vocab_vectorizer(v_ext)

# for demonstration purposes a boolean co-occurrence within sliding window of size 10 is used
# 10 represents sentence co-occurrence, a size of 110 would, e.g., be paragraph co-occurrence
window_size = 5

tcm_ext = create_tcm(iterator_ext, vectorizer_ext
                     ,skip_grams_window = window_size
                     ,weights = rep(1, window_size)
                     ,binary_cooccurence = TRUE
                    )
#add marginal probabilities in diagonal (by default only upper triangle of tcm is created)
diag(tcm_ext) = attributes(tcm_ext)$word_count

# get number of sliding windows that serve as virtual documents, i.e. n_doc_tcm argument
n_skip_gram_windows = sum(sapply(tokens, function(x) {length(x)}))
```

`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.

```
model = Collocations$new(vocabulary = NULL, collocation_count_min = 50, pmi_min = 5, gensim_min = 0,
                         lfmd_min = -Inf, llr_min = 0, sep = "_")
model$partial_fit(it, ...)
model$fit(it, n_iter = 1, ...)
model$transform(it)
model$prune(pmi_min = 5, gensim_min = 0, lfmd_min = -Inf, llr_min = 0)
model$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, llr_min = 0)` filter out non-relevant phrases with low score. User can do it directly by modifying `collocation_stat` object.

Arguments

model A `Collocation` model object

n_iter number of iteration over data

pmi_min, gensim_min, lfmd_min, llr_min minimal scores of the corresponding statistics in order to collapse tokens into collocation:

- pointwise mutual information
- "gensim" scores - https://radimrehurek.com/gensim/models/phrases.html adapted from word2vec paper
- log-frequency biased mutual dependency
- Dunning's logarithm of the ratio between the likelihoods of the hypotheses of dependence and independence

See http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.11.8101&rep=rep1&type=pdf, http://www.aclweb.org/anthology/I05-1050 for details. Also see data in `model$collocation_stat` for better intuition

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

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

it2 = model$transform(it)
v2 = create_vocabulary(it2)
v2 = prune_vocabulary(v2, term_count_min = 5)
# check what phrases model has learned
setdiff(v2$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")
```

`combine_vocabularies` *Combines multiple vocabularies into one*

Description

Combines multiple vocabularies into one

Usage

```
combine_vocabularies(..., combine_stopwords = function(x)
  unique(unlist(lapply(x, attr, which = "stopwords"), use.names = FALSE)),
  combine_ngram = function(x) attr(x[[1]], "ngram"),
  combine_sep_ngram = function(x) attr(x[[1]], "sep_ngram"))
```

Arguments

`...`	vocabulary objects created with create_vocabulary.
`combine_stopwords`	function to combine stopwords from input vocabularies. By default we take a union of all stopwords.
`combine_ngram`	function to combine lower and upper boundary for n-grams from input vocabularies. Usually these values should be the same, so we take this parameter from first vocabulary.
`combine_sep_ngram`	function to combine stopwords from input vocabularies. Usually these values should be the same, so we take this parameter from first vocabulary.

Value

`text2vec_vocabulary` see details in create_vocabulary.

`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",
  "RsparseMatrix"), ...)

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

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

Arguments

it	itoken iterator or list of itoken iterators.
vectorizer	function vectorizer function; see vectorizers.
type	character, one of c("dgCMatrix","dgTMatrix").
...	placeholder for additional arguments (not used at the moment). over it.

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
it = token_parallel(movie_review$review[1:N], tolower, word_tokenizer, movie_review$id[1:N])
vectorizer = hash_vectorizer()
dtm = create_dtm(it, 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, skip_grams_window = 5L,
  skip_grams_window_context = c("symmetric", "right", "left"),
  weights = 1/seq_len(skip_grams_window), binary_cooccurence = FALSE,
  ...)

## 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), binary_cooccurence = FALSE,
  ...)

## 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),
  binary_cooccurence = FALSE, ...)
```

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.
`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.
`skip_grams_window_context`	one of `c("symmetric","right","left")` - which context words to use when count co-occurence statistics.
`weights`	weights for context/distant words during co-occurence statistics calculation. By default we are setting `weight = 1 / distance_from_current_word`. Should have length equal to skip_grams_window.
`binary_cooccurence`	`FALSE` by default. If set to `TRUE` then function only counts first appearence of the context word and remaining occurrence are ignored. Useful when creating TCM for evaluation of coherence of topic models. `"symmetric"` by default - take into account `skip_grams_window` left and right.

... placeholder for additional arguments (not used at the moment). 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 = word_tokenizer(tolower(movie_review$review))
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
it = token_parallel(movie_review$review[1:N], tolower, word_tokenizer, movie_review$id[1:N])
v = create_vocabulary(jobs)
vectorizer = vocab_vectorizer(v)
dtm = create_dtm(it, vectorizer, type = 'dgTMatrix')
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 = "_", window_size = 0L)

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

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

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

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

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. **NOTE** that stopwords will be used "as is". This means that if preprocessing function in itoken does some text modification (like stemming), then this preprocessing need to be applied to stopwords before passing them here. See https://github.com/dselivanov/text2vec/issues/228 for example.
`sep_ngram`	`character` a character string to concatenate words in ngrams
`window_size`	`integer` (0 by default). If `window_size > 0` than vocabulary will be created from pseudo-documents which are obtained by virtually splitting each documents into chunks of the length `window_size` by going with sliding window through them. This is useful for creating special statistics which are used for coherence estimation in topic models.
`...`	placeholder for additional arguments (not used at the moment).

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.
- itoken_parallel: collects unique terms and corresponding statistics from iterator.

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

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"

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.

GloVe *re-export rsparse::GloVe*

Description

re-export rsparse::GloVe

Usage

```
GlobalVectors
```

Format

An object of class R6ClassGenerator of length 24.

`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)

ifiles_parallel(file_paths, 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 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.
`...`	other arguments (not used at the moment)

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 'character'
itoken(iterable, preprocessor = identity,
  tokenizer = space_tokenizer, n_chunks = 10,
  progressbar = interactive(), ids = NULL, ...)

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

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

itoken_parallel(iterable, ...)

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

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

## S3 method for class 'list'
itoken_parallel(iterable, n_chunks = 10, ids = NULL,
  ...)
```

Arguments

`iterable`	an object from which to generate an iterator
`...`	arguments passed to other methods
`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.
n_chunks	integer, the number of pieces that object should be divided into. Then each chunk is processed independently (and in case itoken_parallel **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).
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.

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) {
#   lapply(word_tokenizer(x), SnowballC::wordStem, language="en")
# }
it = itoken_parallel(movie_review$review[1:100], n_chunks = 4)
system.time(dtm <- create_dtm(it, hash_vectorizer(2**16), type = 'dgTMatrix'))
```

jsPCA_robust *(numerically robust) Dimension reduction via Jensen-Shannon Divergence & Principal Components*

Description

This function is largely a copy of the repsective function in https://github.com/cpsievert/LDAvis/blob/master/R/createJSON.R however, with a fix to avoid log(0) proposed by Maren-Eckhoff in https://github.com/cpsievert/LDAvis/issues/56

Usage

```
jsPCA_robust(phi)
```

Arguments

phi	matrix, with each row containing the distribution over terms for a topic, with as many rows as there are topics in the model, and as many columns as there are terms in the vocabulary.

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
```

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 = int
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 method = "Wa`
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 while fitting model

n_iter_inference `integer` number iterations used when sampling from converged model for inference. In other words number of samples from distribution after burn-in.

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) `score_previous_check / score_current < 1 + convergence_tol`

Examples

```
library(text2vec)
data("movie_review")
N = 500
tokens = word_tokenizer(tolower(movie_review$review[1:N]))
it = itoken(tokens, ids = movie_review$id[1:N])
v = create_vocabulary(it)
v = prune_vocabulary(v, 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$new(n_topics)
lsa$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 `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. Preferably in dgCMatrix format

n_topics integer desired number of latent topics.

... Arguments to internal functions. Notably useful for fit_transform() - these arguments will be passed to rsparse::soft_svd

Examples

```
data("movie_review")
N = 100
tokens = word_tokenizer(tolower(movie_review$review[1:N]))
dtm = create_dtm(itoken(tokens), hash_vectorizer(2**10))
n_topics = 5
lsa_1 = LatentSemanticAnalysis$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 !inherits(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$id[train_ind]
txt = tolower(movie_review[['review']][train_ind])
names(txt) = ids
tokens = word_tokenizer(txt)
it = itoken(tokens, progressbar = FALSE, ids = ids)
vocab = create_vocabulary(it)
vocab = prune_vocabulary(vocab, term_count_min = 5, doc_proportion_min = 0.02)
dtm = create_dtm(it, vectorizer = vocab_vectorizer(vocab))
n_topic = 10
model = LDA$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 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.
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 Relaxed Word Movers Distance (RWMD) model*

Description

RWMD model can be used to query the "relaxed word movers distance" from a document to a collection of documents. RWMD tries to measure distance between query document and collection of documents by calculating how hard is to transform words from query document into words from each document in collection. For more detail see following article: http://mkusner.github.io/publications/WMD.pdf. However in contrast to the article above we calculate "easiness" of the convertion of one word into another by using **cosine** similarity (but not a euclidean distance). Also here in text2vec we've implemented effiient RWMD using the tricks from the Linear-Complexity Relaxed Word Mover's Distance with GPU Acceleration article.

Usage

```
RelaxedWordMoversDistance

RWMD
```

Format

R6Class object.

Usage

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

```
rwmd = RelaxedWordMoversDistance$new(x, embeddings)
rwmd$sim2(x)
```

Methods

`$new(x, embeddings)` Constructor for RWMD model. `x` - docuent-term matrix which represents collection of documents against which you want to perform queries. `embeddings` - matrix of word embeddings which will be used to calculate similarities between words (each row represents a word vector).

`$sim(x)` calculates similarity from a collection of documents to collection query documents `x`. `x` here is a document-term matrix which represents the set of query documents

`$dist(x)` calculates distance from a collection of documents to collection query documents `x` `x` here is a document-term matrix which represents the set of query documents

Examples

```
## Not run:
library(text2vec)
library(rsparse)
data("movie_review")
tokens = word_tokenizer(tolower(movie_review$review))
v = create_vocabulary(itoken(tokens))
v = prune_vocabulary(v, 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$new(rank = 50, x_max = 10)
wv = glove_model$fit_transform(tcm, n_iter = 5)
# get average of main and context vectors as proposed in GloVe paper
wv = wv + t(glove_model$components)
rwmd_model = RelaxedWordMoversDistance$new(dtm, wv)
rwms = rwmd_model$sim2(dtm[1:10, ])
head(sort(rwms[1, ], decreasing = T))

## 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 *text2vec*

Description

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

Details

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

TfIdf *TfIdf*

Description

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

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-occurence 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.

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

---

tokenizers *Simple tokenization functions for string splitting*

Description

Few simple tokenization functions. For more comprehensive list see `tokenizers` package: https://cran.r-project.org/package=tokenizers. Also check `stringi::stri_split_*`.

Usage

```
word_tokenizer(strings, ...)

char_tokenizer(strings, ...)

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

postag_lemma_tokenizer(strings, udpipe_model, tagger = "default",
  tokenizer = "tokenizer", pos_keep = character(0),
  pos_remove = c("PUNCT", "DET", "ADP", "SYM", "PART", "SCONJ", "CCONJ",
  "AUX", "X", "INTJ"))
```

Arguments

`strings`	`character` vector
`...`	other parameters (usually not used - see source code for details).
`sep`	`character`, `nchar(sep) = 1` - split strings by this character.
`xptr`	`logical` tokenize at C++ level - could speed-up by 15-50%.
`udpipe_model`	- udpipe model, can be loaded with `?udpipe::udpipe_load_model`
`tagger`	`"default"` - tagger parameter as per `?udpipe::udpipe_annotate` docs.
`tokenizer`	`"tokenizer"` - tokenizer parameter as per `?udpipe::udpipe_annotate` docs.
`pos_keep`	`character(0)` specifies which tokens to keep. `character(0)` means to keep all of them
`pos_remove`	`c("PUNCT","DET","ADP","SYM","PART","SCONJ","CCONJ","AUX","X","INTJ")` - which tokens to remove. `character(0)` is equal to not remove any.

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, " ")
```

`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

`vocabulary`	`text2vec_vocabulary` object, see create_vocabulary.
`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 `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