Package 'mlvocab'

April 13, 2018

Title Vocabulary and Corpus Preprocessing for Natural Language Pipelines

Version 0.0.1

Description Utilities for preprocessing of text corpora into data structures suitable for natural language models: integer sequences or matrices, vocabulary embedding matrices, term-doc, doc-term, term co-occurrence matrices etc. All functions allow for full or partial hashing of the terms in the vocabulary.

Depends R (>= 3.4.0)

License GPL-3

Encoding UTF-8

Imports Rcpp (>= 0.12), Matrix, digest (>= 0.6.8), sparsepp (>= 0.2.0)

LinkingTo Rcpp, digest (>= 0.6.8), sparsepp (>= 0.2.0)

Suggests testthat, knitr

LazyData true

SystemRequirements C++11

BugReports https://github.com/vspinu/mlvocab/issues

URL https://github.com/vspinu/mlvocab/

RoxygenNote 6.0.1

NeedsCompilation yes

Author Vitalie Spinu [aut, cre]

Maintainer Vitalie Spinu <spinuvit@gmail.com>

Repository CRAN

Date/Publication 2018-04-13 08:50:01 UTC

R topics documented:

`term_indices` *Convert text to integer indices*

Description

Convert text to integer indices

Usage

```
tiseq(corpus, vocab, keep_unknown = nbuckets > 0, nbuckets = attr(vocab,
  "nbuckets"), reverse = FALSE)

timat(corpus, vocab, maxlen = 100, pad_right = TRUE, trunc_right = TRUE,
  keep_unknown = nbuckets > 0, nbuckets = attr(vocab, "nbuckets"),
  reverse = FALSE)
```

Arguments

`corpus`	text corpus
`vocab`	data frame produced by `vocab()` or `vocab_update()`
`keep_unknown`	logical. If `TRUE`, preserve unknowns in the output sequences.
`nbuckets`	integer. How many buckets to hash unknowns into.
`reverse`	logical. Should each sequence be reversed in the final output? Reversion happens after `pad_right` and `trunc_right` have been applied to the original text sequence. Default `FALSE`.
`maxlen`	integer. Maximum length of each sequence.
`pad_right`	logical. Should 0-padding of shorter than `maxlen` sequences happen on the right? Default `TRUE`.
`trunc_right`	logical. Should truncation of longer than `maxlen` sequences happen on the right? Default `TRUE`.

Value

`tiseq()` returns a list of integer vectors, `timat()` returns an integer matrix, one row per sequence.

Examples

```
corpus <- list(a = c("The", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog"),
               b = c("the", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog",
                     "the", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog"))
v <- vocab(corpus["b"]) # "The" is unknown
v
tiseq(corpus, v)
tiseq(corpus, v, keep_unknown = TRUE)
tiseq(corpus, v, nbuckets = 1)
```

```
tiseq(corpus, v, nbuckets = 3)

timat(corpus, v, maxlen = 12)
timat(corpus, v, maxlen = 12, keep_unknown = TRUE)
timat(corpus, v, maxlen = 12, nbuckets = 1)
timat(corpus, v, maxlen = 12, nbuckets = 1, reverse = TRUE)
timat(corpus, v, maxlen = 12, pad_right = FALSE, nbuckets = 1)
timat(corpus, v, maxlen = 12, trunc_right = FALSE, nbuckets = 1)
```

`term_matrices` *Term-document and term-cooccurrence matrices*

Description

These functions compute or update various term-counts of a corpus with flexible output specification.

- default weights for the context window `["a" "b" "c" "d" "e"]` a b c d e 1.00 0.50 0.33 0.25 0.20

Usage

```
dtm(corpus, vocab = NULL, ngram = attr(vocab, "ngram"),
  nbuckets = attr(vocab, "nbuckets"), output = c("triplet", "column", "row",
  "df"))

tdm(corpus, vocab = NULL, ngram = attr(vocab, "ngram"),
  nbuckets = attr(vocab, "nbuckets"), output = c("triplet", "column", "row",
  "df"))

tcm(corpus, vocab = NULL, window_size = 5,
  window_weights = 1/seq.int(window_size), context = c("symmetric", "right",
  "left"), ngram = attr(vocab, "ngram"), nbuckets = attr(vocab, "nbuckets"),
  output = c("triplet", "column", "row", "df"))
```

Arguments

`corpus`	a list of character vectors
`vocab`	a `data.frame` produced by an early call to `vocab()`. When `vocab` is `NULL` and `nbuckets` is `NULL` or `0`, the vocabulary is first computed from corpus. When `nbuckets > 0` and `vocab` is `NULL` the result matrix will consist of buckets only.
`ngram`	an integer vector of the form `[ngram_min, ngram_max]`. Defaults to the `ngram` settings used during the creation of `vocab`. Explicitly providing this parameter should rarely be needed.
`nbuckets`	number of unknown buckets
`output`	one of "triplet", "column", "row", "df" or an unambiguous abbreviation thereof. First three options return the corresponding sparse matrices from Matrix package, "df" results in a triplet `data.frame`.

window_size	sliding window size used for co-occurrence computation. In this implementation the window includes the context word; thus, window_size == 1 will result in 0 co-occurrence matrix. This convention allows for consistent weighting schemes across different values of ngram_min and ngram_max.
window_weights	vector of weights which are superimposed on the sliding window. First element is a weight for distance 0 (aka context word itself), second for distance 1 etc. First weight doesn't play any role for ngram_max == 1, see details. window_weights is recycled to length window_size if needed. It can be a string naming a function or a function which accepts one argument, window_size, and returns a window_weights vector. Defaults to [1, 1/2, ..., 1/window_size].
context	when "symmetric", matrix entries (i, j) and (j, i) are the same and represent coocurence of terms i and j within window_size. When "right", entry (i, j) represents coocurence of the term j on the right side of i. When "left", entry (i, j) represents the coocurence of the termjon the left of termi‘.

Details

- for ngram=c(1L, 3L) a a_b a_b_c b b_c b_c_d c c_d c_d_e d d_e e 1.00 0.75 0.61 0.50 0.42 0.36 0.33 0.29 0.26 0.25 0.22 0.20
- for ngram=c(2L, 3L) a_b a_b_c b_c b_c_d c_d c_d_e d_e 0.75 0.61 0.42 0.36 0.29 0.26 0.22

---

tfidf *Tfidf re-weighting of* dtm *and* tdm *matrices*

Description

Tfidf re-weighting of dtm and tdm matrices

Usage

```
tfidf(mat, vocab, norm = c("l1", "l2", "none"), sublinear_tf = FALSE,
  extra_df_count = 1)
```

Arguments

mat	output of dtm() or tdm() function
vocab	output of vocab() or vocab_update()
norm	normalization to apply for each document. Either "l1", "l2" or "none"
sublinear_tf	when TRUE use 1 + log(tf) instead of the raw tf
extra_df_count	add this number to the document count; as if all terms in the vocabulary have been seen at least in this many documents.

Examples

```
corpus <- list(a = c("The", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog"),
               b = c("the", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog",
                     "the", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog"))
v <- vocab(corpus, c(1, 2), " ")
dtm <- dtm(corpus, v)
tfidf(dtm, v)
tdm <- tdm(corpus, v)
tfidf(tdm, v)
```

---

vocab *Build and manipulate vocabularies*

Description

`vocab()` creates a vocabulry from a text corpus; `vocab_update()` and `vocab_prune()`, respectively, update and prune an existing vocabulary.

`vocab_embed()` subsets a (commonly large) pre-trained word-vector matrix into a smaller, one vector per term, embedding matrix.

`vocab_embed()` is commonly used in conjunction with sequence generators (`timat()` and `tiseq()`). When a term in a corpus is not present in a vocabulary (aka unknown), it is hashed into one of the `nbuckets` buckets. Embeddings which are hashed into same bucket are averaged to produce the embedding for that bucket. Maximum number of embeddings to average per bucket is controled with `max_in_bucket` parameter.

Similarly, when a term from the vocabulary is not present in the embedding matrix (aka missing) `max_in_bucket` embeddings are averaged to produce the missing embedding. Different buckets are used for "missing" and "unknown" embeddings because `nbuckets` can be 0.

Usage

```
vocab(corpus, ngram = c(1, 1), ngram_sep = "_")

vocab_update(vocab, corpus)

vocab_prune(vocab, max_terms = Inf, term_count_min = 1L,
  term_count_max = Inf, doc_proportion_min = 0, doc_proportion_max = 1,
  doc_count_min = 1L, doc_count_max = Inf, nbuckets = attr(vocab,
  "nbuckets"))

vocab_embed(vocab, embeddings, nbuckets = attr(vocab, "nbuckets"),
  max_in_bucket = 30)
```

Arguments

`corpus`	list of character vectors

ngram	a vector of length 2 of the form `c(min_ngram, max_ngram)` or a singleton `max_ngram` which is equivalent to `c(1L, max_ngram)`.
ngram_sep	separator to link terms within ngrams.
vocab	`data.frame` obtained from a call to `vocab()`.
max_terms	max number of terms to preserve
term_count_min	keep terms occurring at *least* this many times over all docs
term_count_max	keep terms occurring at *most* this many times over all docs
doc_count_min, doc_proportion_min	keep terms appearing in at *least* this many docs
doc_count_max, doc_proportion_max	keep terms appearing in at *most* this many docs
nbuckets	How many unknown buckets to create along the remaining terms of the pruned `vocab`. All pruned terms will be hashed into this many buckets and the corresponding statistics (`term_count` and `doc_count`) updated.
embeddings	embeddings matrix. The terms dimension must be named. If both `colnames()` and `rownames()` are non-null, dimension with more elements is considered term-dimension.
max_in_bucket	At most this many embedding vectors will be averaged into each unknown or missing bucket (see details). Lower number results in faster processing. For large `nbuckets` this number might not be reached due to the finiteness of the `embeddings` vocabulary, or even result in `0` embeddings being hashed into a bucket producing `[0 0 ...]` embeddings for some buckets.

Examples

```
corpus <-
   list(a = c("The", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog"),
        b = c("the", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog",
              "the", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog"))

vocab(corpus)
vocab(corpus, ngram = 3)
vocab(corpus, ngram = c(2, 3))

v <- vocab(corpus)

extra_corpus <- list(extras = c("apples", "oranges"))
v <- vocab_update(v, extra_corpus)
v

vocab_prune(v, max_terms = 7)
vocab_prune(v, term_count_min = 2)
vocab_prune(v, max_terms = 7, nbuckets = 2)

v2 <- vocab_prune(v, max_terms = 7, nbuckets = 2)
enames <- c("the", "quick", "brown", "fox", "jumps")
emat <- matrix(rnorm(50), nrow = 5,
               dimnames = list(enames, NULL))
```

```
vocab_embed(v2, emat)
vocab_embed(v2, t(emat)) # automatic detection of the orientation

vembs <- vocab_embed(v2, emat)
all(vembs[enames, ] == emat[enames, ])
```

Index