Package 'doc2vec'

May 8, 2026

Type Package

Title Distributed Representations of Sentences, Documents and Topics

Version 0.2.2

Maintainer Jan Wijffels <jwijffels@bnosac.be>

Description Learn vector representations of sentences, paragraphs or documents by using the 'Paragraph Vector' algorithms,
namely the distributed bag of words ('PV-DBOW') and the distributed memory ('PV-DM') model.
The techniques in the package are detailed in the paper ``Distributed Representations of Sentences and Documents'' by Mikolov et al. (2014), available at <doi:10.48550/arXiv.1405.4053>.
The package also provides an implementation to cluster documents based on these embedding using a technique called top2vec.
Top2vec finds clusters in text documents by combining techniques to embed documents and words and density-based clustering.
It does this by embedding documents in the semantic space as defined by the 'doc2vec' algorithm. Next it maps
these document embeddings to a lower-dimensional space using the 'Uniform Manifold Approximation and Projection' (UMAP) clustering algorithm
and finds dense areas in that space using a 'Hierarchical Density-
Based Clustering' technique (HDBSCAN). These dense
areas are the topic clusters which can be represented by the corresponding topic vector which is an aggregate of the
document embeddings of the documents which are part of that topic cluster. In the same semantic space similar words can
be found which are representative of the topic.
More details can be found in the paper 'Top2Vec: Distributed Representations of Topics' by D. Angelov available at <doi:10.48550/arXiv.2008.09470>.

URL https://github.com/bnosac/doc2vec

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

Depends R (>= 2.10)

Imports Rcpp (>= 0.11.5), stats, utils

LinkingTo Rcpp

Suggests tokenizers.bpe, word2vec (>= 0.3.3), uwot, dbscan, udpipe (>= 0.8)

NeedsCompilation yes

Author Jan Wijffels [aut, cre, cph] (R wrapper),
BNOSAC [cph] (R wrapper),
hiyijian [ctb, cph] (Code in src/doc2vec)

Repository CRAN

Date/Publication 2025-11-27 09:50:02 UTC

Contents

`as.matrix.paragraph2vec` *Get the document or word vectors of a paragraph2vec model*

Description

Get the document or word vectors of a paragraph2vec model as a dense matrix.

Usage

```
## S3 method for class 'paragraph2vec'
as.matrix(
  x,
  which = c("docs", "words"),
  normalize = TRUE,
  encoding = "UTF-8",
  ...
)
```

Arguments

x	a paragraph2vec model as returned by paragraph2vec or read.paragraph2vec
which	either one of 'docs' or 'words'
normalize	logical indicating to normalize the embeddings. Defaults to TRUE.
encoding	set the encoding of the row names to the specified encoding. Defaults to 'UTF-8'.
...	not used

Value

a matrix with the document or word vectors where the rownames are the documents or words upon which the model was trained

See Also

paragraph2vec, read.paragraph2vec

Examples

```
library(tokenizers.bpe)
data(belgium_parliament, package = "tokenizers.bpe")
x <- subset(belgium_parliament, language %in% "french")
x <- subset(x, nchar(text) > 0 & txt_count_words(text) < 1000)

model <- paragraph2vec(x = x, type = "PV-DM",   dim = 15,  iter = 5)

model <- paragraph2vec(x = x, type = "PV-DBOW", dim = 100, iter = 20)


embedding <- as.matrix(model, which = "docs")
embedding <- as.matrix(model, which = "words")
embedding <- as.matrix(model, which = "docs", normalize = FALSE)
embedding <- as.matrix(model, which = "words", normalize = FALSE)
```

---

be_parliament_2020 *Corpus with Questions asked in the Belgium Federal Parliament in 2020*

Description

The dataset was extracted from http://data.dekamer.be and contains questions asked by members in the Belgium Federal parliament in 2020.
The Questions are in Dutch and French and contains 6059 text fragments.

The dataset contains the following information:

- doc_id: an identifier
- text_nl: the question itself in Dutch
- text_fr: the question itself in French

Source

data is provided by http://www.dekamer.be in the public domain (CC0).

Examples

```
data(be_parliament_2020)
str(be_parliament_2020)
```

---

paragraph2vec *Train a paragraph2vec also known as doc2vec model on text*

Description

Construct a paragraph2vec model on text. The algorithm is explained at https://arxiv.org/pdf/1405.4053.pdf. People also refer to this model as doc2vec.
The model is an extension to the word2vec algorithm, where an additional vector for every paragraph is added directly in the training.

Usage

```
paragraph2vec(
  x,
  type = c("PV-DBOW", "PV-DM"),
  dim = 50,
  window = ifelse(type == "PV-DM", 5L, 10L),
  iter = 5L,
  lr = 0.05,
  hs = FALSE,
  negative = 5L,
  sample = 0.001,
  min_count = 5L,
  threads = 1L,
  encoding = "UTF-8",
  embeddings = matrix(nrow = 0, ncol = dim),
  ...
)
```

Arguments

x	a data.frame with columns doc_id and text or the path to the file on disk containing training data. Note that the text column should be of type character, should contain less than 1000 words where space or tab is used as a word separator and that the text should not contain newline characters as these are considered document delimiters. The doc_id should not contain spaces.
type	character string with the type of algorithm to use, either one of • 'PV-DM': Distributed Memory paragraph vectors • 'PV-DBOW': Distributed Bag Of Words paragraph vectors Defaults to 'PV-DBOW'.
dim	dimension of the word and paragraph vectors. Defaults to 50.
window	skip length between words. Defaults to 10 for PV-DM and 5 for PV-DBOW
iter	number of training iterations. Defaults to 20.
lr	initial learning rate also known as alpha. Defaults to 0.05
hs	logical indicating to use hierarchical softmax instead of negative sampling. Defaults to FALSE indicating to do negative sampling.
negative	integer with the number of negative samples. Only used in case hs is set to FALSE
sample	threshold for occurrence of words. Defaults to 0.001
min_count	integer indicating the number of time a word should occur to be considered as part of the training vocabulary. Defaults to 5.
threads	number of CPU threads to use. Defaults to 1.
encoding	the encoding of x and stopwords. Defaults to 'UTF-8'. Calculating the model always starts from files allowing to build a model on large corpora. The encoding argument is passed on to file when writing x to hard disk in case you provided it as a data.frame.
embeddings	optionally a matrix with pretrained word embeddings which will be used to initialise the word embedding space with (transfer learning). The rownames of this matrix should consist of words. Only words overlapping with the vocabulary extracted from x will be used.
...	further arguments passed on to the C++ function paragraph2vec_train - for expert use only

Value

an object of class paragraph2vec_trained which is a list with elements

- model: a Rcpp pointer to the model
- data: a list with elements file: the training data used, n (the number of words in the training data), n_vocabulary (number of words in the vocabulary) and n_docs (number of documents)
- control: a list of the training arguments used, namely min_count, dim, window, iter, lr, skipgram, hs, negative, sample

References

https://arxiv.org/pdf/1405.4053.pdf, https://groups.google.com/g/word2vec-toolkit/c/Q49FIrNOQRo/m/J6KG8mUj45sJ

See Also

predict.paragraph2vec, as.matrix.paragraph2vec

Examples

```
library(tokenizers.bpe)
## Take data and standardise it a bit
data(belgium_parliament, package = "tokenizers.bpe")
str(belgium_parliament)
x <- subset(belgium_parliament, language %in% "french")
x$text   <- tolower(x$text)
x$text   <- gsub("[^[:alpha:]]", " ", x$text)
x$text   <- gsub("[[:space:]]+", " ", x$text)
x$text   <- trimws(x$text)
x$nwords <- txt_count_words(x$text)
x <- subset(x, nwords < 1000 & nchar(text) > 0)

## Build the model
model <- paragraph2vec(x = x, type = "PV-DM",   dim = 15,  iter = 5)

model <- paragraph2vec(x = x, type = "PV-DBOW", dim = 100, iter = 20)

str(model)
embedding <- as.matrix(model, which = "words")
embedding <- as.matrix(model, which = "docs")
head(embedding)

## Get vocabulary
vocab <- summary(model, type = "vocabulary",  which = "docs")
vocab <- summary(model, type = "vocabulary",  which = "words")


## Transfer learning using existing word embeddings
library(word2vec)
w2v   <- word2vec(x$text, dim = 50, type = "cbow", iter = 20, min_count = 5)
emb   <- as.matrix(w2v)
model <- paragraph2vec(x = x, dim = 50, type = "PV-DM", iter = 20, min_count = 5,
                       embeddings = emb)


## Transfer learning - proof of concept without learning (iter=0, set to higher to learn)
emb       <- matrix(rnorm(30), nrow = 2, dimnames = list(c("en", "met")))
model     <- paragraph2vec(x = x, type = "PV-DM",   dim = 15,  iter = 0, embeddings = emb)
embedding <- as.matrix(model, which = "words", normalize = FALSE)
embedding[c("en", "met"), ]
emb
```

paragraph2vec_similarity *Similarity between document / word vectors as used in paragraph2vec*

Description

The similarity between document / word vectors is defined as the inner product of the vector elements

Usage

```
paragraph2vec_similarity(x, y, top_n = +Inf)
```

Arguments

x	a matrix with embeddings where the rownames of the matrix provide the label of the term
y	a matrix with embeddings where the rownames of the matrix provide the label of the term
top_n	integer indicating to return only the top n most similar terms from y for each row of x. If top_n is supplied, a data.frame will be returned with only the highest similarities between x and y instead of all pairwise similarities

Value

By default, the function returns a similarity matrix between the rows of x and the rows of y. The similarity between row i of x and row j of y is found in cell [i, j] of the returned similarity matrix. If top_n is provided, the return value is a data.frame with columns term1, term2, similarity and rank indicating the similarity between the provided terms in x and y ordered from high to low similarity and keeping only the top_n most similar records.

See Also

paragraph2vec

Examples

```
x <- matrix(rnorm(6), nrow = 2, ncol = 3)
rownames(x) <- c("word1", "word2")
y <- matrix(rnorm(15), nrow = 5, ncol = 3)
rownames(y) <- c("doc1", "doc2", "doc3", "doc4", "doc5")

paragraph2vec_similarity(x, y)
paragraph2vec_similarity(x, y, top_n = 1)
paragraph2vec_similarity(x, y, top_n = 2)
paragraph2vec_similarity(x, y, top_n = +Inf)
paragraph2vec_similarity(y, y)
paragraph2vec_similarity(y, y, top_n = 1)
```

```
paragraph2vec_similarity(y, y, top_n = 2)
paragraph2vec_similarity(y, y, top_n = +Inf)
```

`predict.paragraph2vec` *Predict functionalities for a paragraph2vec model*

Description

Use the paragraph2vec model to

- get the embedding of documents, sentences or words
- find the nearest documents/words which are similar to either a set of documents, words or a set of sentences containing words

Usage

```
## S3 method for class 'paragraph2vec'
predict(
  object,
  newdata,
  type = c("embedding", "nearest"),
  which = c("docs", "words", "doc2doc", "word2doc", "word2word", "sent2doc"),
  top_n = 10L,
  encoding = "UTF-8",
  normalize = TRUE,
  ...
)
```

Arguments

`object`	a paragraph2vec model as returned by `paragraph2vec` or `read.paragraph2vec`
`newdata`	either a character vector of words, a character vector of doc_id's or a list of sentences where the list elements are words part of the model dictionary. What needs to be provided depends on the argument you provide in `which`. See the examples.
`type`	either 'embedding' or 'nearest' to get the embeddings or to find the closest text items. Defaults to 'nearest'.
`which`	either one of 'docs', 'words', 'doc2doc', 'word2doc', 'word2word' or 'sent2doc' where • 'docs' or 'words' can be chosen if `type` is set to 'embedding' to indicate that `newdata` contains either doc_id's or words • 'doc2doc', 'word2doc', 'word2word', 'sent2doc' can be chosen if `type` is set to 'nearest' indicating to extract respectively the closest document to a document (doc2doc), the closest document to a word (word2doc), the closest word to a word (word2word) or the closest document to sentences (sent2doc).

top_n	show only the top n nearest neighbours. Defaults to 10, with a maximum value of 100. Only used for type 'nearest'.
encoding	set the encoding of the text elements to the specified encoding. Defaults to 'UTF-8'.
normalize	logical indicating to normalize the embeddings. Defaults to TRUE. Only used for type 'embedding'.
...	not used

Value

depending on the type, you get a different output:

- for type nearest: returns a list of data.frames with columns term1, term2, similarity and rank indicating the elements which are closest to the provided newdata
- for type embedding: a matrix of embeddings of the words/documents or sentences provided in newdata, rownames are either taken from the words/documents or list names of the sentences. The matrix has always the same number of rows as the length of newdata, possibly with NA values if the word/doc_id is not part of the dictionary

See the examples.

See Also

paragraph2vec, read.paragraph2vec

Examples

```
library(tokenizers.bpe)
data(belgium_parliament, package = "tokenizers.bpe")
x <- belgium_parliament
x <- subset(x, language %in% "dutch")
x <- subset(x, nchar(text) > 0 & txt_count_words(text) < 1000)
x$doc_id <- sprintf("doc_%s", 1:nrow(x))
x$text   <- tolower(x$text)
x$text   <- gsub("[^[:alpha:]]", " ", x$text)
x$text   <- gsub("[[:space:]]+", " ", x$text)
x$text   <- trimws(x$text)

## Build model
model <- paragraph2vec(x = x, type = "PV-DM",   dim = 15,  iter = 5)

model <- paragraph2vec(x = x, type = "PV-DBOW", dim = 100, iter = 20)


sentences <- list(
  example = c("geld", "diabetes"),
  hi = c("geld", "diabetes", "koning"),
  test = c("geld"),
  nothing = character(),
  repr = c("geld", "diabetes", "koning"))
```

```
## Get embeddings (type =  'embedding')
predict(model, newdata = c("geld", "koning", "unknownword", NA, "</s>", ""),
               type = "embedding", which = "words")
predict(model, newdata = c("doc_1", "doc_10", "unknowndoc", NA, "</s>"),
               type = "embedding", which = "docs")
predict(model, sentences, type = "embedding")

## Get most similar items (type =  'nearest')
predict(model, newdata = c("doc_1", "doc_10"), type = "nearest", which = "doc2doc")
predict(model, newdata = c("geld", "koning"), type = "nearest", which = "word2doc")
predict(model, newdata = c("geld", "koning"), type = "nearest", which = "word2word")
predict(model, newdata = sentences, type = "nearest", which = "sent2doc", top_n = 7)

## Similar way on extracting similarities
emb <- predict(model, sentences, type = "embedding")
emb_docs <- as.matrix(model, type = "docs")
paragraph2vec_similarity(emb, emb_docs, top_n = 3)
```

`read.paragraph2vec` *Read a binary paragraph2vec model from disk*

Description

Read a binary paragraph2vec model from disk

Usage

```
read.paragraph2vec(file)
```

Arguments

`file`	the path to the model file

Value

an object of class paragraph2vec which is a list with elements

- model: a Rcpp pointer to the model
- model_path: the path to the model on disk
- dim: the dimension of the embedding matrix

Examples

```
library(tokenizers.bpe)
data(belgium_parliament, package = "tokenizers.bpe")
x <- subset(belgium_parliament, language %in% "french")
x <- subset(x, nchar(text) > 0 & txt_count_words(text) < 1000)
```

```
model <- paragraph2vec(x = x, type = "PV-DM",   dim = 100, iter = 20)
model <- paragraph2vec(x = x, type = "PV-DBOW", dim = 100, iter = 20)


path <- "mymodel.bin"

write.paragraph2vec(model, file = path)
model <- read.paragraph2vec(file = path)

vocab <- summary(model, type = "vocabulary", which = "docs")
vocab <- summary(model, type = "vocabulary", which = "words")
embedding <- as.matrix(model, which = "docs")
embedding <- as.matrix(model, which = "words")
```

`summary.top2vec` *Get summary information of a top2vec model*

Description

Get summary information of a top2vec model. Namely the topic centers and the most similar words to a certain topic

Usage

```
## S3 method for class 'top2vec'
summary(
  object,
  type = c("similarity", "c-tfidf"),
  top_n = 10,
  data = object$data,
  embedding_words = object$embedding$words,
  embedding_docs = object$embedding$docs,
  ...
)
```

Arguments

`object`	an object of class `top2vec` as returned by `top2vec`
`type`	a character string with the type of summary information to extract for the top-words. Either 'similarity' or 'c-tfidf'. The first extracts most similar words to the topic based on semantic similarity, the second by extracting the words with the highest tf-idf score for each topic
`top_n`	integer indicating to find the `top_n` most similar words to a topic

data	a data.frame with columns 'doc_id' and 'text' representing documents. For each topic, the function extracts the most similar documents. And in case type is 'c-tfidf' it get the words with the highest tf-idf scores for each topic.
embedding_words	a matrix of word embeddings to limit the most similar words to. Defaults to the embedding of words from the object
embedding_docs	a matrix of document embeddings to limit the most similar documents to. Defaults to the embedding of words from the object
...	not used

Examples

```
# For an example, look at the documentation of ?top2vec
```

---

top2vec *Distributed Representations of Topics*

Description

Perform text clustering by using semantic embeddings of documents and words to find topics of text documents which are semantically similar.

Usage

```
top2vec(
  x,
 data = data.frame(doc_id = character(), text = character(), stringsAsFactors = FALSE),
  control.umap = list(n_neighbors = 15L, n_components = 5L, metric = "cosine"),
  control.dbscan = list(minPts = 100L),
  control.doc2vec = list(),
  umap = uwot::umap,
  trace = FALSE,
  ...
)
```

Arguments

x	either an object returned by paragraph2vec or a data.frame with columns 'doc_id' and 'text' storing document ids and texts as character vectors or a matrix with document embeddings to cluster or a list with elements docs and words containing document embeddings to cluster and word embeddings for deriving topic summaries
data	optionally, a data.frame with columns 'doc_id' and 'text' representing documents. This dataset is just stored, in order to extract the text of the most similar documents to a topic. If it also contains a field 'text_doc2vec', this will be used to indicate the most relevant topic words by class-based tfidf

control.umap	a list of arguments to pass on to umap for reducing the dimensionality of the embedding space
control.dbscan	a list of arguments to pass on to hdbscan for clustering the reduced embedding space
control.doc2vec	optionally, a list of arguments to pass on to paragraph2vec in case x is a data.frame instead of a doc2vec model trained by paragraph2vec
umap	function to apply UMAP. Defaults to umap, can as well be tumap
trace	logical indicating to print evolution of the algorithm
...	further arguments not used yet

Value

an object of class top2vec which is a list with elements

- embedding: a list of matrices with word and document embeddings
- doc2vec: a doc2vec model
- umap: a matrix of representations of the documents of x
- dbscan: the result of the hdbscan clustering
- data: a data.frame with columns doc_id and text
- size: a vector of frequency statistics of topic occurrence
- k: the number of clusters
- control: a list of control arguments to doc2vec / umap / dbscan

Note

The topic '0' is the noise topic

References

https://arxiv.org/abs/2008.09470

See Also

paragraph2vec

Examples

```
library(word2vec)
library(uwot)
library(dbscan)
data(be_parliament_2020, package = "doc2vec")
x      <- data.frame(doc_id = be_parliament_2020$doc_id,
                     text   = be_parliament_2020$text_nl,
                     stringsAsFactors = FALSE)
x$text <- txt_clean_word2vec(x$text)
```

```
x      <- subset(x, txt_count_words(text) < 1000)
d2v    <- paragraph2vec(x, type = "PV-DBOW", dim = 50,
                        lr = 0.05, iter = 10,
                        window = 15, hs = TRUE, negative = 0,
                        sample = 0.00001, min_count = 5,
                        threads = 1)
# write.paragraph2vec(d2v, "d2v.bin")
# d2v    <- read.paragraph2vec("d2v.bin")
model  <- top2vec(d2v, data = x,
                  control.dbscan = list(minPts = 50),
                 control.umap = list(n_neighbors = 15L, n_components = 4), trace = TRUE)
model  <- top2vec(d2v, data = x,
                  control.dbscan = list(minPts = 50),
                 control.umap = list(n_neighbors = 15L, n_components = 3), umap = tumap,
                  trace = TRUE)

info   <- summary(model, top_n = 7)
info$topwords
info$topdocs
library(udpipe)
info   <- summary(model, top_n = 7, type = "c-tfidf")
info$topwords

## Change the model: reduce doc2vec model to 2D
model  <- update(model, type = "umap",
                 n_neighbors = 100, n_components = 2, metric = "cosine", umap = tumap,
                 trace = TRUE)
info   <- summary(model, top_n = 7)
info$topwords
info$topdocs

## Change the model: have minimum 200 points for the core elements in the hdbscan density
model  <- update(model, type = "hdbscan", minPts = 200, trace = TRUE)
info   <- summary(model, top_n = 7)
info$topwords
info$topdocs



##
## Example on a small sample
##  with unrealistic hyperparameter settings especially regarding dim / iter / n_epochs
##  in order to have a basic example finishing < 5 secs
##

library(uwot)
library(dbscan)
library(word2vec)
data(be_parliament_2020, package = "doc2vec")
x        <- data.frame(doc_id = be_parliament_2020$doc_id,
                       text   = be_parliament_2020$text_nl,
                       stringsAsFactors = FALSE)
x        <- head(x, 1000)
```

```
x$text   <- txt_clean_word2vec(x$text)
x        <- subset(x, txt_count_words(text) < 1000)
d2v      <- paragraph2vec(x, type = "PV-DBOW", dim = 10,
                          lr = 0.05, iter = 0,
                          window = 5, hs = TRUE, negative = 0,
                          sample = 0.00001, min_count = 5)
emb      <- list(docs  = as.matrix(d2v, which = "docs"),
                 words = as.matrix(d2v, which = "words"))
model    <- top2vec(emb,
                    data = x,
                    control.dbscan = list(minPts = 50),
                    control.umap = list(n_neighbors = 15, n_components = 2,
                                        init = "spectral"),
                    umap = tumap, trace = TRUE)
info     <- summary(model, top_n = 7)
print(info, top_n = c(5, 2))
```

`txt_count_words` *Count the number of spaces occurring in text*

Description

The C++ doc2vec functionalities in this package assume words are either separated by a space or tab symbol and that each document contains less than 1000 words.
This function calculates how many words there are in each element of a character vector by counting the number of occurrences of the space or tab symbol.

Usage

```
txt_count_words(x, pattern = "[ \t]", ...)
```

Arguments

`x`	a character vector with text
`pattern`	a text pattern to count which might be contained in `x`. Defaults to either space or tab.
`...`	other arguments, passed on to `gregexpr`

Value

an integer vector of the same length as `x` indicating how many times the pattern is occurring in `x`

Examples

```
x <- c("Count me in.007", "this is a set  of words",
       "more\texamples tabs-and-spaces.only", NA)
txt_count_words(x)
```

`update.top2vec` *Update a Top2vec model*

Description

Update a Top2vec model by updating the UMAP dimension reduction together with the HDBSCAN clustering or update only the HDBSCAN clustering

Usage

```
## S3 method for class 'top2vec'
update(
  object,
  type = c("umap", "hdbscan"),
  umap = object$umap_FUN,
  trace = FALSE,
  ...
)
```

Arguments

`object`	an object of class `top2vec` as returned by `top2vec`
`type`	a character string indicating what to udpate. Either 'umap' or 'hdbscan' where the former (type = 'umap') indicates to update the umap as well as the hdbscan procedure and the latter (type = 'hdbscan') indicates to update only the hdbscan step.
`umap`	see `umap` argument in `top2vec`
`trace`	logical indicating to print evolution of the algorithm
`...`	further arguments either passed on to `hdbscan` in case type is 'hdbscan' or to `umap` in case type is 'umap'

Value

an updated top2vec object

Examples

```
# For an example, look at the documentation of ?top2vec
```

`write.paragraph2vec` *Save a paragraph2vec model to disk*

Description

Save a paragraph2vec model as a binary file to disk

Usage

```
write.paragraph2vec(x, file)
```

Arguments

`x`	an object of class `paragraph2vec` or `paragraph2vec_trained` as returned by `paragraph2vec`
`file`	the path to the file where to store the model

Value

invisibly a logical if the resulting file exists and has been written on your hard disk

See Also

`paragraph2vec`

Examples

```
library(tokenizers.bpe)
data(belgium_parliament, package = "tokenizers.bpe")
x <- subset(belgium_parliament, language %in% "french")
x <- subset(x, nchar(text) > 0 & txt_count_words(text) < 1000)


model <- paragraph2vec(x = x, type = "PV-DM",   dim = 100, iter = 20)
model <- paragraph2vec(x = x, type = "PV-DBOW", dim = 100, iter = 20)


path <- "mymodel.bin"

write.paragraph2vec(model, file = path)
model <- read.paragraph2vec(file = path)

vocab <- summary(model, type = "vocabulary", which = "docs")
vocab <- summary(model, type = "vocabulary", which = "words")
embedding <- as.matrix(model, which = "docs")
embedding <- as.matrix(model, which = "words")
```

Index