Package 'sbert'

November 17, 2026

Type Package

Title Topic Modeling with 'Sentence-BERT' Embeddings

Version 0.6.0

Description Builds semantic topic models from 'Sentence-BERT' embeddings (Reimers and Gurevych, 2019) <doi:10.18653/v1/D19-1410>. Sentence embeddings are computed in several modes with models run on the Open Neural Network Exchange ('ONNX') Runtime, including multilingual and long-context encoders and a fast static model. Topics are labeled with class-based term weighting that can be retuned without refitting. Candidate topic counts and document granularities (whole documents, sentences, clauses, or phrases) can be compared side by side, and fitted models assign topics to new documents, give soft memberships and per-document topic mixtures. The package offers coherence and diversity metrics and visualizations.

License MIT + file LICENSE

URL https://sonsoles.me/sbert/, https://github.com/sonsoleslp/sbert

BugReports https://github.com/sonsoleslp/sbert/issues

Encoding UTF-8

Language en-US

LazyData true

Depends R (>= 4.1.0)

Imports digest, grDevices, graphics, onnxr (>= 0.1.2), parallel, tok (>= 0.2.3), stats, tools, utils

Suggests SnowballC, jsonlite, knitr, rmarkdown, testthat (>= 3.0.0), withr

SystemRequirements ONNX Runtime (>= 1.25.1)

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.1.1

NeedsCompilation no

Author Mohammed Saqr [aut, cph] (ORCID:
<https://orcid.org/0000-0001-5881-3109>),
Sonsoles López-Pernas [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0002-9621-1392>)

Maintainer Sonsoles López-Pernas <sonsoles.lopez@uef.fi>

Repository CRAN

Date/Publication 2026-11-17 21:00:02 UTC

Contents

sbert-package . . . 3
abbreviations . . . 5
as.data.frame.sbert_topic_corpus . . . 6
as.data.frame.sbert_topic_hierarchy . . . 7
as.data.frame.sbert_topic_model . . . 8
as.data.frame.sbert_topic_sweep . . . 9
blend . . . 10
cache_dir . . . 11
cache_size . . . 12
clean_corpus . . . 13
coherence . . . 15
compare_topics . . . 16
content_ratio . . . 19
covid . . . 20
dedupe . . . 21
encode . . . 22
feedback_embeddings . . . 24
feedback_translations . . . 25
fitted.sbert_topic_sweep . . . 26
install_runtime . . . 27
keywords . . . 28
load_custom . . . 29
load_model . . . 31
models . . . 33
model_download . . . 34
model_remove . . . 35
model_status . . . 36
name_topics . . . 37
plot.sbert_topic_hierarchy . . . 38
plot.sbert_topic_model . . . 39
plot.sbert_topic_sweep . . . 41
pool . . . 42
predict.sbert_topic_model . . . 44
print.sbert_model . . . 45
print.sbert_topic_corpus . . . 46

print.sbert_topic_hierarchy . . . 47
print.sbert_topic_model . . . 48
print.sbert_topic_sweep . . . 49
print.summary.sbert_topic_hierarchy . . . 49
print.summary.sbert_topic_model . . . 50
print.summary.sbert_topic_sweep . . . 51
reduce_topics . . . 52
representatives . . . 53
segment . . . 55
stop_words . . . 56
strip_list_markers . . . 58
summary.sbert_topic_hierarchy . . . 59
summary.sbert_topic_model . . . 60
summary.sbert_topic_sweep . . . 61
terms.sbert_topic_model . . . 62
topics . . . 64
topic_corpus . . . 68
topic_diversity . . . 70
topic_gamma . . . 71
topic_hierarchy . . . 73
topic_membership . . . 74
topic_palette . . . 76
topic_similarity . . . 77
topic_sizes . . . 78

Index **80**

`sbert-package` *sbert: Topic modeling with Sentence-BERT embeddings*

Description

`sbert` fits topic models of texts from their sentence embeddings. A sentence embedding places a text in a space where texts with similar meaning lie close together, whatever words they use (Reimers and Gurevych 2019), so topics found in that space group texts by what they mean rather than by the words they share. Each topic is described by its most typical texts and by its distinctive terms, scored with class-based TF-IDF (Grootendorst 2022). Topics can be fitted to whole documents or to their sentences, clauses, or phrases, compared across numbers of topics by coherence and diversity, and used to assign new texts and give each document a mixture of topics.

The embeddings come from pretrained models, each fixed to one version (see `models()`). A model is downloaded only when it is first needed, with `model_download()` or after a prompt in an interactive session, and is stored in the user cache (`cache_dir()`); nothing is downloaded when the package is installed or loaded. The bundled feedback_embeddings let every topic-model function run without any download.

Models and embeddings

Texts are tokenized by the 'tok' package and encoded on the ONNX Runtime through 'onnxr'; pooling, normalization, and all of the topic modeling are done in R. Each model is fixed to one revision and its files are checked by SHA-256, so every user encodes with the same model.

- `install_runtime()` installs the ONNX Runtime library.
- `models()`, `model_status()`, `model_download()`, `load_model()`, and `load_custom()` list, download, and load models; `cache_dir()`, `cache_size()`, and `model_remove()` manage the cache.
- `encode()` embeds texts; `pool()` pools token embeddings; `topic_similarity()` gives cosine similarities between embeddings; `blend()` combines segment embeddings with their document's.

Preparing a corpus

- `clean_corpus()`, `strip_list_markers()`, and `content_ratio()` clean the texts.
- `dedupe()` collapses repeated texts and counts them.
- `segment()` splits documents into sentences, clauses, or phrases; `abbreviations()` lists the abbreviations that do not end a sentence.
- `topic_corpus()` segments, embeds, and tokenizes a corpus once.

Fitting and choosing topic models

- `topics()` fits a topic model.
- `compare_topics()` compares numbers of topics and segmentation levels.
- `topic_hierarchy()` and `reduce_topics()` relate and merge topics.

The fits are deterministic: k-means starts from fixed farthest-point centers, so no random seed is involved and the same input gives the same topics.

Describing topics

- `topic_sizes()`, terms(), `representatives()`, and as.data.frame() give the tables of a model; `name_topics()` names its topics.
- summary(), `coherence()`, and `topic_diversity()` evaluate it; `keywords()` gives keywords of single documents; `stop_words()` sets the words left out of the terms.
- plot() draws sizes, terms, representatives, a fit report, and a map; `topic_palette()` gives its colors.

Assigning topics to texts

- predict() assigns new texts to topics.
- `topic_membership()` gives soft memberships, and `topic_gamma()` the mixture of topics in each document.

Author(s)

Maintainer: Sonsoles López-Pernas <sonsoles.lopez@uef.fi> (ORCID) [copyright holder]

Authors:

- Sonsoles López-Pernas <sonsoles.lopez@uef.fi> (ORCID) [copyright holder]
- Mohammed Saqr <mohammed.saqr@uef.fi> (ORCID) [copyright holder]

References

Reimers, N. and Gurevych, I. (2019). Sentence-BERT: Sentence embeddings using Siamese BERT-networks. *Proceedings of EMNLP-IJCNLP 2019*, 3982–3992. doi:10.18653/v1/D191410

Grootendorst, M. (2022). BERTopic: Neural topic modeling with a class-based TF-IDF procedure. arXiv:2203.05794.

See Also

Useful links:

- https://sonsoles.me/sbert/
- https://github.com/sonsoleslp/sbert
- Report bugs at https://github.com/sonsoleslp/sbert/issues

`abbreviations` *List the abbreviations protected from sentence splitting*

Description

`abbreviations()` returns the built-in list of abbreviations whose period does not end a sentence, which `segment()` uses by default. A period ends a sentence far more often than not, but abbreviations such as "e.g.", "Fig." or "et al." also end in one, so rule-based sentence splitting needs such a list.

Matching in `segment()` ignores case and starts at a word boundary, so `"ST."` protects `"St. Petersburg"` but not the end of `"cost."`.

To protect further abbreviations, pass an extended vector to the `abbreviations` argument of `segment()`.

Usage

```
abbreviations()
```

Value

A sorted character vector of abbreviations in upper case, each ending in a period.

See Also

`segment()`, which uses the list.

Examples

```
abbreviations()

# "App." is not in the list, so its period ends a sentence ...
segment("See App. B for the items. Then stop.", level = "sentence")

# ... unless it is added.
segment("See App. B for the items. Then stop.", level = "sentence",
        abbreviations = c(abbreviations(), "APP."))
```

---

as.data.frame.sbert_topic_corpus

Convert a prepared topic corpus to a data frame

Description

`as.data.frame()` returns the units of a prepared corpus, one row per unit to be clustered: each document, or each sentence, clause, or phrase when the corpus is segmented. The table shows what `topics()` and `compare_topics()` will model, including the segments a segmentation produced and the columns carried from a data frame input.

Usage

```
## S3 method for class 'sbert_topic_corpus'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

`x`	A corpus prepared by `topic_corpus()`.
`row.names`, `optional`	Ignored; present for compatibility with the generic.
`...`	Ignored.

Value

A base data frame with one row per unit and columns `document_id`, `document_name`, `segment` (segmented corpora only), and `text`, followed by the columns carried from a data frame input.

See Also

`topic_corpus()`, `segment()`.

Examples

```
documents <- data.frame(
  id = c("a", "b"),
  text = c("Cats chase mice. Stocks and bonds trade.",
           "Dogs chase balls. Markets price shares.")
)
prepared <- topic_corpus(
  documents, column = "text", segment = "sentence",
  embeddings = rbind(c(1, 0), c(0, 1), c(0.9, 0.1), c(0.1, 0.9))
)
as.data.frame(prepared)
```

---

as.data.frame.sbert_topic_hierarchy

Convert a topic hierarchy to a data frame

Description

`as.data.frame()` returns the merge table of a topic hierarchy: one row per merge of the hierarchical clustering of the topic centers, in the order in which the merges happen. Merges at a small height join topics whose centers are nearly parallel; such topics are candidates for `reduce_topics()`.

Usage

```
## S3 method for class 'sbert_topic_hierarchy'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

`x`	A hierarchy built by `topic_hierarchy()`.
`row.names, optional`	Ignored; present for compatibility with the generic.
`...`	Ignored.

Value

A base data frame with one row per merge and columns `step`, `height` (the cosine distance at which the two branches join), `left`, and `right` (the branches, as `"topic <id> (<label>)"` or `"merge <step>"`).

See Also

summary() for the nearest neighbor of each topic; plot() for the dendrogram.

Examples

```
corpus <- head(dedupe(feedback_translations, column = "translation"), 600)
topic_model <- topics(corpus, n_topics = 6, column = "translation",
                      embeddings = feedback_embeddings, label = "terms")
as.data.frame(topic_hierarchy(topic_model))
```

---

as.data.frame.sbert_topic_model

Convert a topic model to a data frame

Description

`as.data.frame()` returns one of the tables of a fitted topic model as a base data frame. The default, `what = "documents"`, gives one row per modeled unit with its topic, so that the topic of every document (or segment) can be joined to other data, filtered with `subset()`, or tabulated.

The other tables are the topic sizes (`"topics"`), the stored terms (`"terms"`), and the stored representative units (`"representatives"`), as kept when the model was fitted. To compute terms under other settings, use terms(); to rank representatives in another way, use `representatives()`.

Usage

```
## S3 method for class 'sbert_topic_model'
as.data.frame(
  x,
  row.names = NULL,
  optional = FALSE,
  ...,
  what = c("documents", "topics", "terms", "representatives")
)
```

Arguments

`x`	A topic model fitted by `topics()`.
`row.names, optional`	Ignored; present for compatibility with the generic.
`...`	Ignored.
`what`	Which table: `"documents"` (default), `"topics"`, `"terms"`, or `"representatives"`.

Value

A base data frame:

- `"documents"`: one row per modeled unit, with `document_id`, `document_name`, `segment` (segmented fits only), `text`, `topic`, `label`, `distance` (cosine distance to the topic center), and the columns carried from a data frame input.

- "topics": one row per topic, with topic, label, n_documents, proportion, and withinss (within-topic sum of squares).
- "terms": one row per topic and term, with topic, label, term, rank, score, and frequency.
- "representatives": one row per topic and representative unit, with topic, label, rank, document_id, document_name, segment (segmented fits only), text, and distance.

See Also

topic_sizes(), terms(), representatives(), predict().

Examples

```
corpus <- head(dedupe(feedback_translations, column = "translation"), 600)
topic_model <- topics(corpus, n_topics = 6, column = "translation",
                      embeddings = feedback_embeddings)
head(as.data.frame(topic_model))
as.data.frame(topic_model, what = "topics")
subset(as.data.frame(topic_model), topic == 6 & n > 2)
```

---

as.data.frame.sbert_topic_sweep

Convert a topic-count comparison to a data frame

Description

as.data.frame() returns the table of a comparison of candidate numbers of topics as a plain data frame, without the fitted models, for reporting or for joining with other results.

Usage

```
## S3 method for class 'sbert_topic_sweep'
as.data.frame(x, ...)
```

Arguments

x	A comparison returned by compare_topics().
...	Ignored.

Value

A base data frame with one row per candidate and columns n_topics, coherence, topic_diversity, explained, and converged, preceded by segment when several levels were compared. The attributes measure and coherence_weight are kept.

See Also

compare_topics(); summary().

Examples

```
corpus <- head(dedupe(feedback_translations, column = "translation"), 600)
comparison <- compare_topics(corpus, n_topics = c(4, 6),
                             column = "translation",
                             embeddings = feedback_embeddings)
as.data.frame(comparison)
```

`blend` *Blend segment embeddings with their document context*

Description

`blend()` computes context-aware embeddings of segments: it combines what each segment says beyond its parent document with the direction of the document itself. A short segment is often ambiguous on its own ("It was too hard."), and identical segments can occur in different documents with different meanings; the parent document supplies the missing context.

Each segment vector `u` is split into the part along its unit-length document vector `d` and the residual orthogonal to it; the blend keeps `alpha` of the residual and `1 - alpha` of the document direction, scaled to unit length:

$$m = \text{normalize}\left(\alpha\left(u - (u \cdot d)\, d\right) + (1 - \alpha)\, d\right)$$

with `u` and `d` normalized first. `alpha = 0` returns the document vector, `alpha = 1` only what the segment adds to it. Identical segments from different documents therefore receive different vectors. A segment that points exactly along its document receives the document vector.

The result, a matrix with one unit-length row per segment, replaces the segment embeddings in the `embeddings` argument of `topics()`, `representatives()` and predict(). To cluster segments by their own meaning alone, pass the plain segment embeddings from `encode()` instead.

Usage

```
blend(
  segments,
  documents = NULL,
  model = NULL,
  alpha = 0.5,
  embeddings = NULL,
  document_embeddings = NULL,
  batch_size = 32L
)
```

Arguments

`segments`	A data frame from `segment()`, or any data frame with a numeric `document_id` and a `text` column, one row per segment.
`documents`	The character vector of documents that was segmented; `document_id` indexes it. Needed unless both embedding matrices are given.

model	A model from load_model(), a pinned model name, or NULL (default) for the default model; ignored when embeddings are given.
alpha	Weight of the segment's residual, in [0, 1] (default 0.5).
embeddings	Segment embeddings, one row per row of segments, or NULL (default) to encode them. Give together with document_embeddings.
document_embeddings	Document embeddings, one row per document, or NULL (default) to encode documents.
batch_size	Texts per model call when encoding (default 32).

Value

A numeric matrix with one unit-length row per row of segments and the columns of the embeddings.

Conditions

sbert_bad_input when only one of embeddings and document_embeddings is given, or neither they nor documents; when encoding, the conditions of encode().

See Also

segment() for the segments; encode() for the embeddings; topics() to fit topics on the blended vectors.

Examples

```
segments <- segment("The model works. It is fast.", level = "sentence")
segment_vectors <- matrix(c(1, 0, 0.6, 0.8), nrow = 2, byrow = TRUE)
document_vector <- matrix(c(0, 1), nrow = 1)
blend(segments, embeddings = segment_vectors,
      document_embeddings = document_vector)

# alpha = 0 returns the document vector for every segment.
blend(segments, alpha = 0, embeddings = segment_vectors,
      document_embeddings = document_vector)
```

---

cache_dir *Locate the model cache*

Description

cache_dir() returns the path of the model cache: the user cache directory that R designates for the package, where downloaded models are stored once per user, outside any project. Every function that installs, verifies, loads, measures or removes a model uses this location unless given another cache_dir.

The path is the one given by `tools::R_user_dir("sbert", which = "cache")`. `cache_dir()` only computes it: it creates no directory and does not check whether one exists.

To see what the cache holds, use `model_status()` for one model and `cache_size()` for the disk space used.

Usage

```
cache_dir()
```

Value

A character string: the path of the cache directory.

See Also

`model_status()`, `cache_size()`, `model_download()` and `model_remove()`, which take this path as `cache_dir`.

Examples

```
cache_dir()
```

---

cache_size *Measure the disk space of the model cache*

Description

`cache_size()` measures the disk space of the model cache: it sums the sizes of all regular files beneath a cache root, including downloaded models and the files of custom models. A pinned model takes from about 31 MB to 1.3 GB of disk (the `size_mb` column of `models()`), and the cache keeps every model and custom model downloaded until it is removed, so this total is the basis for deciding what to remove.

It returns the total size in bytes. To list the files of one model, use `model_status()`; to delete one, use `model_remove()`.

Usage

```
cache_size(cache_dir = default_cache_dir())
```

Arguments

`cache_dir`	Cache root (default the package cache directory, `cache_dir()`).

Value

A number: the total size in bytes, or `0` when the directory does not exist or is empty.

See Also

cache_dir() for the location; model_status() for one model's files; model_remove() to free space.

Examples

```
cache_size(tempdir())
cache_size(file.path(tempdir(), "no-such-cache"))
```

---

clean_corpus *Clean a corpus before encoding*

Description

clean_corpus() removes material without topical meaning from a corpus before encoding: markup, links, citation numbers, list markers, and broken characters. Text exported from web pages, databases, and documents carries such material; left in place, it shifts the embeddings, appears among the topic terms, and can form topics of its own. Cleaning the text once, before encoding, means that the encoder, the topic terms, and the representative texts all see the same clean text.

It always repairs broken characters and removes markup, links, e-mail addresses, DOIs, and bracketed citation numbers. It removes list markers and section numbering by default, and on request numbers, Roman numerals, and the matches of patterns of one's own (remove). Words, author citations, dates, decimals, and numbers within sentences are kept. Documents left empty, or with a share of letters below min_content (see content_ratio()), are dropped. For a data frame, the dropped rows are removed whole, so other columns stay aligned with the text. It returns the cleaned texts that are kept or, for a data frame, the kept rows with column cleaned.

The numbers, roman_numerals, and section_numbers arguments of topics(), by contrast, filter only the topic terms and leave the text unchanged.

Usage

```
clean_corpus(
  text,
  column = NULL,
  list_markers = TRUE,
  section_numbers = TRUE,
  numbers = FALSE,
  roman_numerals = FALSE,
  remove = NULL,
  min_content = 0
)
```

Arguments

text	A character vector, or a data frame together with column.
column	Name of the text column when text is a data frame; NULL (default) for a character vector.
list_markers	Remove list and enumeration markers (default TRUE); see strip_list_markers().
section_numbers	Remove section numbering such as 1.2.3 and II.2 (default TRUE).
numbers	Remove stand-alone numbers such as years and counts, keeping words with digits such as covid19 (default FALSE).
roman_numerals	Remove stand-alone Roman numerals of two or more letters, such as ii and xiv (default FALSE).
remove	Optional character vector of Perl-compatible regular expressions, matched case-insensitively; each match is replaced by a space. Default NULL.
min_content	Minimum share of letters for a document to be kept, between 0 and 1 (default 0, keep every non-empty document).

Value

For a character vector, the cleaned texts that are kept. For a data frame, the rows that are kept, with column cleaned.

Conditions

An error of class sbert_bad_input when column is given for a character vector.

See Also

strip_list_markers() and content_ratio(), which it uses; dedupe() to collapse repeated texts after cleaning.

Examples

```
clean_corpus(c(
  "1. The directive entered into force in 2020.",
  "OJ No L 297, 24.11.1979, p. 1."
), min_content = 0.5)

# Remove a reference form of one's own corpus
clean_corpus("as set out in OJ No L 313 it applies.",
             remove = "OJ No L?\\s*\\d+")
```

coherence *Score the coherence of topics*

Description

coherence() scores the coherence of each topic of a fitted model: whether the top terms of the topic tend to occur in the same texts. A topic whose terms co-occur is more likely to be interpretable as one subject than a topic whose terms rarely appear together, and coherence is the usual intrinsic measure of topic quality, computed from the corpus itself without external judgments.

Each topic is scored from its top n_terms terms and the co-occurrence of those terms in the modeled units (documents, or segments of a segmented model). Two measures are available: normalized pointwise mutual information (NPMI; Bouma 2009), the mean over term pairs of how much more often two terms co-occur than expected, bounded between -1 and 1; and UMass coherence (Mimno et al. 2011), the mean log ratio of the number of units holding both terms of a pair to the number holding the higher-ranked one, usually negative. For both, larger values are more coherent. The result is a table with one row per topic; the mean over topics is kept with it, and is what compare_topics() and summary() report.

Coherence is computed from words, while the topics are formed from embeddings, so a topic of short texts that share meaning but not vocabulary can score low. It complements reading the representative texts and does not replace it.

Usage

```
coherence(
  object,
  measure = c("npmi", "umass"),
  n_terms = 10L,
  smoothing = 1,
  token_lists = NULL,
  cores = 1L,
  weight = c("none", "size")
)
```

Arguments

object	A topic model fitted by topics().
measure	"npmi" (default) or "umass".
n_terms	Top terms per topic to score (default 10), capped at the terms a topic has.
smoothing	Count added to each co-occurrence count for UMass (default 1).
token_lists	Optional precomputed token lists, one per modeled unit, as held by a topic_corpus(); NULL (default) tokenizes the text. The scores are the same either way.
cores	Number of forked worker processes for tokenization (default 1; Unix-alikes only). Results do not depend on it.
weight	How topic scores are averaged into the mean: "none" (default), every topic equally, or "size", each topic in proportion to its number of units, so that a very small topic of near-identical texts cannot move the mean.

Value

A base data frame with one row per topic and columns `topic`, `label`, `measure`, `n_terms` (terms scored), and `coherence` (`NA` for a topic with fewer than two terms). The mean over the scored topics is attached as the attribute `mean_coherence` (`NA` when no topic can be scored) and the averaging as `weight`; summary() prints the mean.

References

Bouma, G. (2009). Normalized (pointwise) mutual information in collocation extraction. *Proceedings of GSCL*, 31–40.

Mimno, D., Wallach, H. M., Talley, E., Leenders, M., and McCallum, A. (2011). Optimizing semantic coherence in topic models. *Proceedings of EMNLP 2011*, 262–272.

See Also

`topic_diversity()`; summary() for the model report; `compare_topics()` to compare numbers of topics.

Examples

```
corpus <- head(dedupe(feedback_translations, column = "translation"), 600)
topic_model <- topics(corpus, n_topics = 6, column = "translation",
                      embeddings = feedback_embeddings)
coherence(topic_model)
coherence(topic_model, measure = "umass", weight = "size")
```

`compare_topics` *Compare numbers of topics and segmentation levels*

Description

`compare_topics()` fits `topics()` for every candidate number of topics, and for every segmentation level in `segment`, on one shared set of embeddings, and reports measures of the quality of each fit. The number of topics, like the unit of analysis (the whole document or its sentences, clauses, or phrases), sets the granularity of a topic model and has no single correct value: fewer topics give broad themes, more topics give narrower ones, until some of them split a theme into near-duplicates.

Three measures are reported per candidate: the mean coherence of the topics' terms (`coherence()`), the diversity of the terms across topics (`topic_diversity()`), and the share of the embedding variance that lies between topics. Coherence tends to fall and between-topic variance to rise as the number of topics grows, so a useful candidate is often the count after which coherence stops improving rather than the count with the highest coherence. Comparing candidates on such measures is a better basis for the choice than fixing a number in advance. The function compares; it does not choose.

The result is a table of class `sbert_topic_sweep` with one row per candidate. The fitted candidates are kept, so a chosen one is taken out with fitted() without refitting.

The measures are a guide; the choice should also rest on reading the representative texts of the candidate models.

Usage

```
compare_topics(
  text,
  n_topics = c(5L, 10L, 15L, 20L, 25L, 30L),
  column = NULL,
  model = NULL,
  embeddings = NULL,
  measure = c("npmi", "umass"),
  n_terms = 10L,
  n_representatives = 1L,
  keep_models = TRUE,
  batch_size = 32L,
  cores = 1L,
  segment = c("document", "sentence", "clause", "phrase"),
  max_tokens = NULL,
  merge_below = 0L,
  min_content = 0,
  coherence_weight = c("none", "size"),
  ...
)
```

Arguments

text	Documents: a character vector, a data frame together with column, or a corpus prepared by topic_corpus().
n_topics	Candidate numbers of topics: distinct integers of at least 2 and below the number of modeled units. Default c(5, 10, 15, 20, 25, 30).
column	Name of the text column when text is a data frame. The other columns are carried into the document table. Rows whose text is missing, blank, or the placeholder "[No abstract available]" are dropped, and so are their rows of embeddings. NULL (default) for a character vector.
model	The encoder: a model from load_model(), a pinned model name, or NULL (default) for the default model. Not used when embeddings are given.
embeddings	Optional precomputed embedding matrix, one row per modeled unit. When several segment levels are compared, a list of matrices named by level. Default NULL.
measure	Coherence measure: "npmi" (default) or "umass".
n_terms	Top terms per topic used for coherence and diversity (default 10).
n_representatives	Representative units stored per topic in each candidate (default 1).
keep_models	Whether to keep the fitted candidates (default TRUE). With FALSE, only the table is kept and fitted() is not available.
batch_size	Texts encoded per model call (default 32).
cores	Number of forked worker processes for tokenization and for fitting the candidates (default 1; Unix-alikes only). Results do not depend on it.

segment
: Unit or units of analysis, as in topics(): one or more of "document" (default), "sentence", "clause", and "phrase". With several levels, every level is fitted for every candidate.

max_tokens, merge_below, min_content
: Segmentation options passed to segment(), for segmented fits only: the maximum segment length (in the model's tokens, or in words with precomputed embeddings; default NULL, no limit), the number of words below which a segment is joined to its neighbor (default 0), and the minimum share of letters for a segment to be kept (default 0; see content_ratio()).

coherence_weight
: How the coherences of a candidate's topics are averaged: "none" (default), every topic equally, or "size", each topic in proportion to its number of units, so a very small topic cannot decide the comparison.

...
: Further arguments to topics(), for example stop_words or weighting.

Value

An object of class sbert_topic_sweep: a data frame with one row per candidate and columns n_topics, coherence (the mean over the topics that can be scored, averaged as coherence_weight says; NA when no topic can be scored), topic_diversity, explained (the between-topic share of the variance), and converged, preceded by segment when several levels are compared. The coherence measure and the weighting are kept as the attributes measure and coherence_weight. summary() gives the most coherent candidate per level, plot() draws the measures, and fitted() returns a fitted candidate.

Conditions

Errors of class sbert_bad_input when fewer than three units are modeled, a candidate is not below the number of units, a list of embeddings is given for a single level, several levels are given with a prepared corpus or with embeddings that are not a list named by level; and the conditions of topics() for each candidate. Non-convergence of a candidate is recorded in the converged column.

See Also

topics() to fit one model; coherence() and topic_diversity() for the measures; topic_hierarchy() and reduce_topics() to merge the topics of a fit.

Examples

```
corpus <- head(dedupe(feedback_translations, column = "translation"), 600)
comparison <- compare_topics(corpus, n_topics = c(4, 6, 8),
                             column = "translation",
                             embeddings = feedback_embeddings)
comparison
summary(comparison)
fitted(comparison, n_topics = 6)

# Documents against sentences (one embedding matrix per level)
```

```
documents <- c(
  "Cats chase mice. Stocks and bonds trade.",
  "Dogs chase balls. Markets price shares.",
  "Kittens nap in sunshine. Banks report profit."
)
levels <- compare_topics(
  documents,
  n_topics = 2,
  segment = c("document", "sentence"),
  embeddings = list(
    document = rbind(c(0.7, 0.7), c(0.6, 0.8), c(0.8, 0.6)),
    sentence = rbind(
      c(1, 0), c(0, 1), c(0.9, 0.1),
      c(0.1, 0.9), c(0.95, 0.05), c(0.05, 0.95)
    )
  ),
  n_terms = 2
)
levels
```

`content_ratio` *Compute the share of letters in text*

Description

`content_ratio()` computes the share of letters among the non-space characters of each element of `text`. The share separates prose from other units without relying on a language: sentences score close to one, while reference lists, citations, tables of numbers, and identifiers, which consist mostly of digits and punctuation and form spurious topics of their own when they are clustered, score low.

It returns one value between 0 and 1 per element and is the measure behind the `min_content` filters of `clean_corpus()`, `segment()`, and `topics()`.

Usage

```
content_ratio(text)
```

Arguments

`text`	A character vector.

Value

A numeric vector of the same length as `text`, with values between 0 and 1; an empty or all-space element scores 0.

See Also

`clean_corpus()` and `segment()`, which filter on it.

Examples

```
content_ratio(c("A real sentence here.", "OJ No L 297, 24.11.1979, p. 1."))
```

`covid` *COVID-19 research abstracts*

Description

`covid` is a corpus of abstracts of peer-reviewed research on COVID-19, drawn from a bibliographic database export and covering the effects of the pandemic on education, children, schools, and society. The abstracts are longer and more technical than the messages of feedback_translations, and every record has a publication year, so the corpus serves for topic models of long documents, for segmented fits, and for following topics across the years of the pandemic.

Usage

```
covid
```

Format

A data frame with 4,170 rows and 2 columns:

Year Integer publication year, from 2020 to 2024.

Abstract The article abstract. 323 records carry the placeholder `"[No abstract available]"`; the other 3,847 hold 3,670 distinct abstracts, the longest of 5,285 characters.

Editorial notices (retractions, corrections, and errata) were removed, so every row is a research abstract or a placeholder. `topics()` and `compare_topics()` drop the placeholder rows when given the data frame with `column = "Abstract"`.

Source

Bibliographic database export of COVID-19 research articles (2020-2024), prepared by `data-raw/covid.R`.

See Also

`topics()` with `segment = "sentence"` for long documents; `clean_corpus()`; feedback_translations for a corpus of short texts.

Examples

```
xtabs(~ Year, data = covid)

abstracts <- subset(covid, Abstract != "[No abstract available]")
nrow(abstracts)
with(abstracts, summary(nchar(Abstract)))
```

dedupe *Collapse repeated texts and count them*

Description

`dedupe()` collapses repeated texts: it keeps the first occurrence of every distinct non-blank text, in order of first appearance, and adds its number of occurrences as `n`. Many corpora repeat texts (templates, standard replies, and copied messages); repeated texts carry no new information about the topics, but in a clustering they pull the topic centers toward themselves and inflate the size of the topics they fall in.

Modeling each distinct text once, and carrying how often it occurred, separates what kinds of text there are from how often each kind is used. The result is a data frame with one row per distinct text. The counts can be used as weights after fitting, for example with `topic_sizes(weights = "n")`.

Usage

```
dedupe(text, column = NULL)
```

Arguments

`text`	A character vector, or a data frame together with `column`. Missing and blank texts are dropped.
`column`	Name of the text column when `text` is a data frame; `NULL` (default) for a character vector.

Value

A base data frame with one row per distinct text, in order of first appearance. For a character vector, the columns are `text` and `n`; for a data frame, all its columns, with the values of the row where the text first occurs, followed by `n`.

Conditions

An error of class `sbert_bad_input` when no non-blank text remains, when `column` is missing, not found, or given for a character vector, or when the data frame already has a column `n`.

See Also

`topic_sizes()` to weight topic sizes by the counts; `clean_corpus()` to clean the texts first.

Examples

```
dedupe(c("Very good!", "Try again.", "Very good!", NA, "  "))

corpus <- dedupe(feedback_translations, column = "translation")
head(corpus)
```

encode *Encode texts as sentence embeddings*

Description

`encode()` returns one sentence embedding per text from a pinned model (see `models()`): a point in a vector space learned so that texts with similar meaning lie close together, whatever words they use. Sentence-BERT models (Reimers and Gurevych, 2019) obtain such a vector by running a transformer encoder over the text and pooling its token vectors; the cosine of the angle between two embeddings then measures how close the two texts are in meaning.

Each text is tokenized with the model's own tokenizer, truncated to the model's maximum length, run through the ONNX graph, pooled with the model's rule (`pool()`), and by default scaled to unit length. The static `potion-base-8M` model instead averages looked-up token vectors and needs no ONNX Runtime. With `cache`, embeddings are stored in a file keyed by text and model, so encoding an edited corpus again computes only the new or changed texts. The result is a numeric matrix with one row per text, in input order.

Encoding is the expensive step of a workflow, and every analysis in the package (topics, the assignment of new texts, similarity) starts from its result: encode once and pass the matrix to the `embeddings` argument of `topics()`, `compare_topics()`, `topic_corpus()`, predict(), `representatives()` and `topic_gamma()`. To split long documents into sentences or clauses first, use `segment()`. The bundled feedback_embeddings matrix is the output of `encode()` for 600 feedback messages and lets the examples run without a model.

Usage

```
encode(
  text,
  model = NULL,
  batch_size = 32L,
  normalize = TRUE,
  prefix = NULL,
  sort_by_length = FALSE,
  cache = NULL
)
```

Arguments

`text`	Character vector, one text per element. Empty strings are allowed; `NA` is not. Names become the row names of the result.
`model`	A model from `load_model()` or `load_custom()`, the name of a pinned model from `models()`, or `NULL` (default) for `"all-MiniLM-L6-v2"`. A name is loaded once per session. A model that is not installed is downloaded only with consent: an interactive prompt, or `options(sbert.download = TRUE)` in a script.
`batch_size`	Number of texts per model call (default 32).
`normalize`	Scale each embedding to unit L2 length (default `TRUE`).

`prefix` Text prepended to every input. `NULL` (default) uses the model's pinned prefix (for example `"query: "` for E5 models); `""` uses none.

`sort_by_length` Batch texts of similar length together, which reduces computation over padding, most of all for segments of varying length (default `FALSE`). The rows keep input order, but the values can differ from an unsorted run by floating-point rounding, enough to change the assignment of a rare unit on a topic boundary.

`cache` `NULL` (default), or the path of an `.rds` file of stored embeddings. Texts already stored for the same model and settings are read from the file; the others are encoded once each (repeats within the call included) and added to it. A file that cannot be read, or was written for another embedding dimension, is rebuilt with a warning.

Value

A numeric matrix with one row per text, in input order, and one column per embedding dimension of the model (the `dimensions` column of `models()`). Row names are `names(text)`. An empty `text` gives a matrix with no rows.

Conditions

`sbert_unknown_model` when `model` names no pinned model; `sbert_model_not_installed` when the model is not installed and its download is declined or not permitted; `sbert_download_failed` or `sbert_cache_invalid` when a permitted download fails or does not verify; `sbert_cache_invalid` when installed files fail verification; `sbert_runtime_missing` when ONNX Runtime is not installed; `sbert_missing_package` when the static model is loaded without jsonlite; `sbert_model_error` when the model cannot be loaded, or inference fails or returns non-finite values. Warning `sbert_cache_rebuilt` when the `cache` file is unusable.

References

Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence embeddings using Siamese BERT-networks. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 3982-3992. doi:10.18653/v1/D191410

See Also

`load_model()` and `models()` for the models; `segment()` to split documents first; `topic_similarity()` to compare embeddings; `topics()` to fit topics on them.

Examples

```
# The bundled embeddings are what encode() returns for 600 feedback
# messages: one row per text, one column per model dimension.
dim(feedback_embeddings)
topic_similarity(head(feedback_embeddings, 3))


if (interactive()) {
  model <- load_model()
```

```
  embeddings <- encode(c("A short sentence.", "Another sentence."), model)

  # Segment-level work benefits most, because segment lengths vary widely.
  segments <- with(head(covid, 50), segment(Abstract, level = "clause"))
  embeddings <- with(segments, encode(text, model, sort_by_length = TRUE))

  # Encode once, then re-run after editing the corpus: only changed rows cost
  # anything the second time.
  store <- file.path(tempdir(), "covid-embeddings.rds")
  embeddings <- with(covid, encode(Abstract, model, cache = store))
  embeddings <- with(covid, encode(Abstract, model, cache = store))
  unlink(store)
}
```

`feedback_embeddings` *Precomputed embeddings of 600 feedback messages*

Description

`feedback_embeddings` is a matrix of Sentence-BERT embeddings of the first 600 distinct English translations of feedback_translations, in order of first appearance. They let topic models be fitted, compared, and inspected without downloading a model, and they are the embeddings used by the examples and articles of the package.

Row `i` is the embedding of row `i` of `head(dedupe(feedback_translations, column = "translation"), 600)`. Pass the matrix as `embeddings =` to `topics()`, `compare_topics()`, or `topic_corpus()` together with that data frame.

Usage

```
feedback_embeddings
```

Format

A numeric matrix with 600 rows (messages) and 384 columns (embedding dimensions). Rows are L2-normalized and rounded to four decimals.

Source

Encoded with the pinned `sentence-transformers/all-MiniLM-L6-v2` model (revision `1110a243fdf4706b3f48f1d95db1`) by `data-raw/feedback_embeddings.R`. Encoding the same messages with `encode()` gives the same embeddings up to the rounding.

See Also

feedback_translations, `topics()`, `encode()`.

Examples

```
dim(feedback_embeddings)

corpus <- head(dedupe(feedback_translations, column = "translation"), 600)
topic_model <- topics(corpus, n_topics = 6, column = "translation",
                      embeddings = feedback_embeddings)
topic_sizes(topic_model, weights = "n")
```

---

feedback_translations *AI-generated mathematics feedback with English translations*

Description

feedback_translations is a dataset of AI-generated feedback messages shown to young learners solving mathematics exercises in an educational application, paired with their English translations. The messages are short instructional hints and encouragements ("Choose the picture...", "Try listening to the instruction again.") whose source languages include Czech, Slovak, Polish, German, Hungarian, Romanian, Ukrainian, Russian, Mongolian, and Vietnamese.

The corpus keeps the quirks of real data: translations repeat (templates such as "Try again." recur), some source messages remain in their original language, and a few are identical to their translation. It is the example corpus of the package for embedding, similarity, segmentation, and topic modeling; feedback_embeddings holds precomputed embeddings of its first 600 distinct translations.

Usage

```
feedback_translations
```

Format

A data frame with 8,757 rows and 2 character columns:

feedback The original feedback message in the source language.

translation The English translation of the message.

There are 8,005 distinct translations, of at most 193 characters.

Source

Anonymized export of the AI mathematics feedback of the Levebee educational application (https://www.levebee.com/), December 2025. The messages address learners generically and contain no personal names or identifiers.

See Also

feedback_embeddings for precomputed embeddings; dedupe() to collapse repeated messages; covid for a corpus of longer texts.

Examples

```
head(feedback_translations)

# Repeated messages collapse to one row with their count
head(dedupe(feedback_translations, column = "translation"))

# Split the first five translations into sentences
with(head(feedback_translations, 5), segment(translation, level = "sentence"))
```

`fitted.sbert_topic_sweep` *Extract a fitted model from a topic-count comparison*

Description

`fitted()` returns one candidate model from a `compare_topics()` comparison, identical to a `topics()` fit with the same settings, without fitting it again. `compare_topics()` fits a model for every candidate number of topics and, by default, keeps them, so once a candidate has been chosen from the comparison its model can be taken out directly.

Usage

```
## S3 method for class 'sbert_topic_sweep'
fitted(object, n_topics, segment = NULL, ...)
```

Arguments

`object`	A comparison returned by `compare_topics()` with `keep_models = TRUE` (the default).
`n_topics`	The number of topics of the model to return; one of the candidates.
`segment`	The segment level of the model to return, when the comparison spans several levels; `NULL` (default) for a single-level comparison.
`...`	Ignored.

Value

A topic model of class `sbert_topic_model`, as returned by `topics()`.

Conditions

Errors of class `sbert_no_fitted_model` when the comparison kept no models or holds no model for `n_topics`, and `sbert_bad_input` when `segment` is given for a single-level comparison or missing for a comparison of several levels.

See Also

`compare_topics()`; summary() for the most coherent candidate.

Examples

```
corpus <- head(dedupe(feedback_translations, column = "translation"), 600)
comparison <- compare_topics(corpus, n_topics = c(4, 6),
                             column = "translation",
                             embeddings = feedback_embeddings)
topic_model <- fitted(comparison, n_topics = 6)
topic_sizes(topic_model, weights = "n")
```

`install_runtime` *Install ONNX Runtime*

Description

`install_runtime()` downloads and installs ONNX Runtime, the native inference library on which transformer models run, through `onnxr::onnx_install()`: the portable CPU build, a CUDA build where supported, or the build chosen by automatic detection. The CRAN package onnxr provides the R interface but leaves the library itself, a large platform-specific binary that is not part of R, to be installed on request.

Nothing is installed when the package is installed or loaded; this function is the only path. Run it once per R library before `load_model()` or `load_custom()`. The static `potion-base-8M` model does not need it. Model files are installed separately with `model_download()`.

Usage

```
install_runtime(cuda = NULL)
```

Arguments

cuda	`FALSE` for the CPU runtime, `TRUE` to request CUDA on supported systems, or `NULL` (default) for automatic detection; passed to `onnxr::onnx_install()`.

Value

The value of `onnxr::onnx_install()`, invisibly.

See Also

`load_model()`, which needs the runtime; `model_download()` for the model files; `onnxr::onnx_is_installed()` to check the installation.

Examples

```
# Whether the ONNX Runtime is already available; nothing is installed here.
onnxr::onnx_is_installed()


if (interactive()) {
  install_runtime()
```

```
}
```

`keywords` *Extract the keywords of documents*

Description

`keywords()` extracts the keywords of each document: the words and short phrases that best express what it is about. Embedding-based keyword extraction compares the meaning of each candidate phrase with the meaning of the whole document, and a phrase whose embedding is close to the document's embedding summarizes it well; because picking the closest phrases alone tends to return near-synonyms, the selection also rewards phrases that differ from those already chosen, by maximal marginal relevance (Carbonell and Goldstein 1998), as in KeyBERT (Grootendorst 2020).

The candidates are the words and phrases of up to `ngrams` consecutive words of each document after stop words are removed; the documents and the candidates are encoded with the same model, and up to `n` keywords per document are selected. The result is a data frame with one row per keyword. `keywords()` describes single documents; to describe topics, use terms(), which compares the words of a topic with those of the other topics.

`keywords()` encodes text, so it needs a downloaded model.

Usage

```
keywords(
  text,
  model = NULL,
  n = 10L,
  ngrams = 2L,
  topic_diversity = 0.3,
  stop_words = default_stop_words(),
  min_token_length = 3L,
  batch_size = 32L
)
```

Arguments

`text`	Documents: a character vector. Names, if any, are carried into `document_name`.
`model`	The encoder: a model from `load_model()`, a pinned model name, or `NULL` (default) for the default model.
`n`	Maximum number of keywords per document (default 10).
`ngrams`	Maximum length of a candidate phrase, in words (default 2).
`topic_diversity`	Trade-off between relevance and diversity, in `[0, 1)`: 0 ranks by similarity to the document alone, and larger values penalize candidates similar to those already chosen (default 0.3).

`stop_words` Words excluded from the candidates; default `stop_words()`.

`min_token_length` Minimum length of a candidate word, in characters (default 3).

`batch_size` Texts encoded per model call (default 32).

Value

A base data frame with one row per keyword and columns `document_id`, `document_name`, `rank`, `keyword`, and `topic_similarity` (the cosine similarity between the keyword and its document).

Conditions

An error of class `sbert_no_terms` when no candidate is left after stop words and short words are removed; the errors of `load_model()` and `encode()`.

References

Carbonell, J. and Goldstein, J. (1998). The use of MMR, diversity-based reranking for reordering documents and producing summaries. *Proceedings of SIGIR 1998*, 335–336. doi:10.1145/290941.291025

Grootendorst, M. (2020). KeyBERT: Minimal keyword extraction with BERT. doi:10.5281/zenodo.4461265

See Also

terms() for the terms of topics; `stop_words()`; `model_download()` to download a model.

Examples

```
# Pinned models that can encode the text
models()


if (interactive()) {
  keywords(
    "Transition network analysis models learning event sequences.",
    n = 5
  )
}
```

`load_custom` *Load a Hugging Face embedding model outside the pinned list*

Description

load_custom() loads a Hugging Face embedding model that is not in the pinned list, on a "trust on first use" basis. Many embedding models on Hugging Face publish an ONNX export of their encoder and a tokenizer.json; such a model can be used like a pinned one, but its configuration is not vouched for by the package: there is no pinned revision, no package-controlled hash, and no check that its embeddings match the reference implementation. The caller is responsible for the pooling rule, prefix and maximum length.

On first use it resolves the current revision (unless revision is given), downloads the ONNX graph and tokenizer, and records their sizes, hashes and the model's configuration in a local manifest; later loads verify the files against that manifest. A later load with the same revision given uses no network; with revision = NULL every call first asks Hugging Face for the current revision. Pooling and maximum length are read from the repository's Sentence-Transformers configuration when it has one and must be given otherwise; the embedding dimension is read from the ONNX graph.

The result is an sbert_model like the one load_model() returns, so encode(), topics() and every other verb accept it. For the pinned, verified models use load_model().

Usage

```
load_custom(
  id,
  revision = NULL,
  onnx_path = NULL,
  tokenizer_path = "tokenizer.json",
  pooling = NULL,
  prefix = NULL,
  max_length = NULL,
  cache_dir = default_cache_dir(),
  backend = "cpu",
  threads = 1L,
  verify = TRUE,
  quiet = FALSE
)
```

Arguments

id	Hugging Face repository id, for example "thenlper/gte-small".
revision	Commit hash to pin, or NULL (default) to resolve the current revision online on every call. Pin it for offline reuse.
onnx_path	Path of the ONNX graph in the repository, or NULL (default) to use onnx/model.onnx or model.onnx.
tokenizer_path	Path of the tokenizer in the repository (default "tokenizer.json").
pooling	"mean" or "cls", or NULL (default) to read it from the repository's 1_Pooling/config.json.
prefix	Text prepended to every input by encode(). NULL (default) keeps the recorded prefix ("" on first use); "" clears it.
max_length	Maximum word pieces per input, or NULL (default) to read it from sentence_bert_config.json, else 512.

cache_dir	Cache root (default cache_dir()).
backend	ONNX execution backend, as accepted by onnxr::onnx_model() (default "cpu").
threads	Number of inference threads (default 1).
verify	Check recorded sizes and SHA-256 hashes when loading from the cache (default TRUE).
quiet	Suppress the download message and progress (default FALSE).

Value

An object of class sbert_model with the same fields as load_model() returns; its short_name is "custom:<id>".

Conditions

sbert_runtime_missing when ONNX Runtime is not installed; sbert_missing_package when jsonlite is not installed; sbert_download_failed when repository metadata or files cannot be downloaded or the revision cannot be resolved; sbert_model_error when the repository has no ONNX graph at the expected path, the pooling rule cannot be detected and was not given, or the graph lacks a last_hidden_state output or needs unsupported inputs; sbert_cache_invalid when cached files no longer match the recorded manifest.

See Also

load_model() for the pinned models; install_runtime() for ONNX Runtime; encode() to use the model; cache_dir() for where it is stored.

Examples

```
# Custom models are stored under "custom" in this cache root.
cache_dir()


if (interactive()) {
  model <- load_custom("thenlper/gte-small")
  encode(c("one sentence", "another sentence"), model)
}
```

---

load_model *Load an installed pinned model*

Description

load_model() reads an installed pinned model from the cache, verifies its files against the pinned sizes and hashes, and returns the model ready for inference: the tokenizer and the ONNX session (or, for the static potion-base-8M, its embedding table). An embedding model is defined by its network weights, its tokenizer, and the rules applied around them (pooling, input prefix, maximum input length); the package pins each supported model to an immutable Hugging Face revision with known file sizes and SHA-256 hashes, so that the same name always means the same model and the same text always receives the same embedding.

It never downloads anything; the files must first be installed with model_download(), and transformer models also need ONNX Runtime from install_runtime(). The result is an object of class sbert_model.

Most verbs accept a model name and load it themselves, keeping it for the rest of the session, so an explicit load_model() is needed mainly to set backend or threads. For a model outside the pinned list, use load_custom().

Usage

```
load_model(
  model = "all-MiniLM-L6-v2",
  cache_dir = default_cache_dir(),
  backend = "cpu",
  threads = 1L,
  verify = TRUE
)
```

Arguments

model	Name of a pinned model, as listed by models() (default "all-MiniLM-L6-v2").
cache_dir	Cache root (default cache_dir()).
backend	ONNX execution backend: "cpu" (default), "coreml", "cuda", "xnnpack" or "openvino", as accepted by onnxr::onnx_model().
threads	Number of inference threads, or "auto" for the machine's performance-core count (default 1). Results do not depend on the thread count; the gain in speed is modest and bounded by the number of performance cores. The default of one keeps runs within the two-core limit of check environments.
verify	Check file sizes and SHA-256 hashes before loading (default TRUE).

Value

An object of class sbert_model (class c("sbert_static_model", "sbert_model") for potion-base-8M): a list holding the model's identity (id, short_name, revision), its encoding settings (dimension, max_length, pooling, prefix), the backend and threads it runs with, and the loaded tokenizer and model. Pass it to the model argument of encode() and of the topic verbs; its print method shows the identity.

Conditions

`sbert_unknown_model` when `model` names no pinned model; `sbert_model_not_installed` when its files are missing; `sbert_cache_invalid` when they fail size or hash verification; `sbert_runtime_missing` when ONNX Runtime is not installed (transformer models only); `sbert_missing_package` when the static model is loaded without jsonlite; `sbert_model_error` when the tokenizer or model cannot be loaded.

See Also

`models()` for the pinned models; `model_download()` and `install_runtime()` to install them; `model_status()` to check an installation; `load_custom()` for other Hugging Face models; `encode()` to use the model.

Examples

```
# The pinned models that load_model() accepts.
models()


if (interactive()) {
  model <- load_model()
  model
}
```

`models` *List the pinned models*

Description

`models()` lists the pinned embedding models as a table: by default the columns needed to choose a model, and with `detail = TRUE` also the settings the package applies when encoding (pooling rule and input prefix), the inference engine, the license, and the repository and revision. Embedding models differ in the languages they cover, the length of input they read, the size of their vectors, and the disk and time they cost.

The package supports a fixed set of models, each pinned to an immutable Hugging Face revision and verified by file size and SHA-256 hash, so a model name identifies exactly one model. The result has one row per pinned model.

Pass a value of the `model` column to any argument that takes a model, for example `encode(text, model = "bge-small-en-v1.5")`, or to `load_model()` and `model_download()`. For a model outside the list, see `load_custom()`.

Usage

```
models(detail = FALSE)
```

Arguments

detail	FALSE (default) for the columns model, dimensions, max_tokens, languages and size_mb; TRUE adds pooling ("mean" or "cls"), prefix (text prepended by encode()), engine ("onnx" or "static"), license, id and revision.

Value

A base data frame with one row per pinned model and the columns described under detail. size_mb is the download size in megabytes.

See Also

load_model() and model_download() to use a model; model_status() to check an installation; load_custom() for other models.

Examples

```
models()
models(detail = TRUE)
```

---

model_download *Download a pinned model*

Description

model_download() downloads the files of one pinned model into the cache. A pinned model is a fixed set of files (an ONNX graph and a tokenizer, or an embedding table for the static model) at an immutable Hugging Face revision; installing it locally once makes encoding work offline and keeps the model identical across machines and over time.

Only files that are missing or fail verification are fetched, and each is checked against its pinned size and SHA-256 hash before it is kept. The function runs only when called; the package never downloads during installation, loading, examples or tests. It returns the model's directory in the cache, invisibly.

Verbs given a model name that is not installed offer this download themselves (see encode()). ONNX Runtime, needed by transformer models, is installed separately with install_runtime(). For models outside the pinned list, see load_custom().

Usage

```
model_download(
  model = "all-MiniLM-L6-v2",
  cache_dir = default_cache_dir(),
  quiet = FALSE,
  timeout = 600
)
```

Arguments

model	Name of a pinned model, as listed by models() (default "all-MiniLM-L6-v2").
cache_dir	Cache root (default cache_dir()).
quiet	Suppress the start message and download progress (default FALSE).
timeout	Minimum download timeout in seconds (default 600); a larger getOption("timeout") is kept.

Value

The model's directory in the cache, invisibly.

Conditions

sbert_unknown_model when model names no pinned model; sbert_download_failed when a file cannot be downloaded or moved into the cache; sbert_cache_invalid when a downloaded file fails size or hash verification.

See Also

models() for the pinned models; model_status() to check an installation; load_model() to load it; model_remove() to delete it; install_runtime() for ONNX Runtime.

Examples

```
# Report what is already cached; nothing is downloaded here.
model_status(tempdir())


if (interactive()) {
  model_download()
}
```

---

model_remove *Remove a pinned model from the cache*

Description

model_remove() deletes the cached directory of one pinned model at its pinned revision, freeing its disk space. Downloaded models stay in the user cache until removed; a removed model can be installed again at the same pinned revision with model_download(), giving identical files.

Other models and the rest of the cache are left in place, and a model that is not installed is not an error. It returns TRUE, invisibly, when the model's directory no longer exists.

Custom models from load_custom() are stored under the custom subdirectory of the cache and are not removed by this function.

Usage

```
model_remove(cache_dir = default_cache_dir(), model = "all-MiniLM-L6-v2")
```

Arguments

cache_dir	Cache root (default cache_dir()).
model	Name of a pinned model, as listed by models() (default "all-MiniLM-L6-v2").

Value

Invisibly, TRUE when the model's directory no longer exists.

Conditions

sbert_unknown_model when model names no pinned model.

See Also

cache_size() for disk use; model_status() to check a model; model_download() to install it again.

Examples

```
cache <- file.path(tempdir(), "sbert-example-cache")
model_remove(cache)
```

---

model_status *Check the installed files of a model*

Description

model_status() checks the installed files of a pinned model: for each file the model needs, it reports where the file is expected in the cache, whether it exists, and whether its size and SHA-256 hash match the pinned values. A pinned model is usable only when each of its files is present and identical to the file the package was pinned to; size and hash establish that identity, so a truncated download or a modified file is detected before it can produce wrong embeddings.

It reads the cache only; nothing is downloaded or repaired. The result is a data frame with one row per required file.

To install missing or invalid files, use model_download(); to delete a model, model_remove().

Usage

```
model_status(cache_dir = default_cache_dir(), model = "all-MiniLM-L6-v2")
```

Arguments

cache_dir	Cache root (default cache_dir()).
model	Name of a pinned model, as listed by models() (default "all-MiniLM-L6-v2").

Value

A base data frame with one row per required file and columns `file`, `path` (its expected location), `exists`, `valid` (size and hash both match), `expected_bytes`, and `actual_bytes` (`NA` when the file is missing).

Conditions

`sbert_unknown_model` when `model` names no pinned model.

See Also

`model_download()` to install; `model_remove()` to delete; `cache_size()` for disk use; `load_model()`, which runs the same check.

Examples

```
# An empty cache: every file is missing.
model_status(tempdir())
model_status(tempdir(), model = "potion-base-8M")
```

`name_topics` *Name the topics of a fitted model*

Description

`name_topics()` replaces the label of every topic of a fitted model with the names given, in topic order. The labels that `topics()` gives are drawn from the data (the most central text of each topic, or its most distinctive terms); once the representative texts of each topic have been read, a short name for what the topic is about, such as the kind of message it holds, communicates the result better.

The names then appear in every table and figure that refers to a topic, including summary(), `topic_sizes()`, terms(), `representatives()`, predict(), `topic_hierarchy()`, and plot(). The model is not refitted; the result is the topic model with its labels replaced.

`reduce_topics()` drops the names, because merged topics need new ones.

Usage

```
name_topics(object, labels)
```

Arguments

`object`	A topic model fitted by `topics()`.
`labels`	Character vector of distinct, non-empty names, one per topic, in topic order (topic 1 first).

Value

The topic model with its labels replaced; an object of class `sbert_topic_model`.

Conditions

An error of class `sbert_bad_topic_names` when `labels` is not a character vector of distinct, non-empty, non-missing names, one per topic.

See Also

`representatives()` to read each topic before naming it; `topics()`.

Examples

```
text <- c(
  "Cats chase mice", "Dogs chase balls",
  "Stocks and bonds trade", "Markets price shares"
)
embeddings <- rbind(c(1, 0), c(0.9, 0.1), c(0, 1), c(0.1, 0.9))
topic_model <- topics(text, n_topics = 2, embeddings = embeddings)
named <- name_topics(topic_model, c("Pets", "Finance"))
topic_sizes(named)
```

`plot.sbert_topic_hierarchy` *Plot a topic hierarchy as a dendrogram*

Description

`plot()` draws the hierarchy of the topics as a dendrogram with the topic labels as leaves and the cosine distance between topic centers on the vertical axis. Topics joined low in the tree are near-duplicates; topics joined only near the top are distinct from the rest.

Usage

```
## S3 method for class 'sbert_topic_hierarchy'
plot(x, main = "Topic hierarchy", cex = 0.8, ...)
```

Arguments

`x`	A hierarchy built by `topic_hierarchy()`.
`main`	Plot title (default `"Topic hierarchy"`).
`cex`	Size of the leaf labels (default 0.8).
`...`	Passed to the `plot()` method of `stats::hclust()`.

Value

`x`, invisibly. Called for the plot.

See Also

topic_hierarchy(); summary(); reduce_topics().

Examples

```
corpus <- head(dedupe(feedback_translations, column = "translation"), 600)
topic_model <- topics(corpus, n_topics = 6, column = "translation",
                      embeddings = feedback_embeddings, label = "terms")
plot(topic_hierarchy(topic_model))
```

---

plot.sbert_topic_model

Plot a fitted topic model

Description

`plot()` draws one of five base-graphics views of a fitted topic model: how large the topics are, which words distinguish them, which texts exemplify them, and how they lie relative to one another in the embedding space. Together the views support naming the topics and judging whether the chosen number of topics is adequate.

`"sizes"` is a bar chart of the number of units (documents, or segments for a segmented model) in each topic. `"terms"` shows each topic's top terms by class-based TF-IDF score, within-topic probability (beta), or count. `"representatives"` lists each topic's representative texts, those closest to its center. `"fit"` combines both in one row per topic: the representative texts, then the terms by count, TF-IDF and beta. `"map"` places the units in two dimensions by classical multidimensional scaling of their cosine distances, colored by topic and labeled at each topic's mean position; large corpora are thinned to `max_points` by an even, non-random sample within each topic.

The data behind each view are available as tables: topic_sizes(), terms(), representatives(), and `as.data.frame(x, what = "documents")`. To compare models with different numbers of topics, plot a compare_topics() result.

Usage

```
## S3 method for class 'sbert_topic_model'
plot(
  x,
  type = c("sizes", "terms", "representatives", "fit", "map"),
  by = "score",
  topics = NULL,
  per_topic = FALSE,
  n_terms = 8L,
  n_representatives = 5L,
  colors = topic_palette(nrow(x$topics)),
  max_points = 1500L,
  main = NULL,
  ...
)
```

Arguments

x	A fitted model from topics().
type	"sizes" (default), "terms", "representatives", "fit", or "map".
by	For type = "terms", one or more of "score" (default; class-based TF-IDF), "beta" (within-topic word probability) and "frequency" (within-topic count). Several values draw one row per topic with a column per measure.
topics	Topic numbers to draw for "terms", "representatives" and "fit", or NULL (default) for all.
per_topic	For "terms", "representatives" and "fit", draw one figure per topic instead of one grid (default FALSE).
n_terms	Terms per panel for "terms" and "fit" (default 8).
n_representatives	Texts per topic for "representatives" and "fit" (default 5). More than the model stored are recomputed from its stored embeddings, if any.
colors	Topic colors, at least one per topic (default topic_palette()).
max_points	Maximum number of units drawn by "map" (default 1500).
main	Title for "fit" drawn as one figure, or NULL (default) for a title naming the layout.
...	Unused.

Value

x, invisibly. The plot is drawn on the current device.

Conditions

sbert_no_embeddings when type = "map" and the model stores no embeddings; sbert_bad_input when type = "map" and the model has fewer than three units; sbert_plot_too_small when the device is too small for the "fit" grid. Warning sbert_few_representatives when more representatives are requested than stored and the model has no embeddings to recompute them.

See Also

topic_sizes(), terms(), representatives() for the tables behind the views; topic_palette() for the colors; compare_topics() to compare numbers of topics.

Examples

```
text <- c(
  "Cats chase mice", "Dogs chase balls",
  "Stocks and bonds trade", "Markets price shares",
  "Neural nets learn patterns", "Models train on data"
)
embeddings <- rbind(
  c(1, 0, 0), c(0.9, 0.1, 0),
  c(0, 1, 0), c(0.1, 0.9, 0),
  c(0, 0, 1), c(0.05, 0, 0.95)
```

```
)
topic_model <- topics(text, 3, embeddings = embeddings)
plot(topic_model, type = "sizes")
plot(topic_model, type = "terms")
plot(topic_model, type = "terms", by = c("frequency", "score", "beta"),
     topics = 1)
plot(topic_model, type = "representatives")
plot(topic_model, type = "fit")
plot(topic_model, type = "fit", per_topic = TRUE)
plot(topic_model, type = "map")
```

`plot.sbert_topic_sweep` *Plot a topic-count comparison*

Description

`plot()` draws the measures of a topic-count comparison, or the fit report of one of its candidates. With `type = "metrics"` (default) it draws three panels against the number of topics: mean coherence, topic diversity, and the share of the variance between topics, with a dotted line at the most coherent candidate. A comparison of several segment levels draws one line per level, told apart by color, point shape, and line type. The useful signal is usually the count after which coherence stops improving, not the highest value.

`type = "fit"` draws the fit report of plot() for the kept model with `n_topics` topics, one figure per segment level, so that one number of topics can be read across segmentations.

Usage

```
## S3 method for class 'sbert_topic_sweep'
plot(
  x,
  type = c("metrics", "fit"),
  main = "Topic-count comparison",
  n_topics = NULL,
  segment = NULL,
  n_terms = 8L,
  n_representatives = 5L,
  ...
)
```

Arguments

`x`	A comparison returned by `compare_topics()`.
`type`	`"metrics"` (default) or `"fit"`.
`main`	Overall title for `type = "metrics"` (default `"Topic-count comparison"`).
`n_topics`	For `type = "fit"`, the number of topics of the models to draw; required when the comparison holds several candidates. Default `NULL`.

segment — For type = "fit", the segment levels to draw; NULL (default) for every level.

n_terms, n_representatives — For type = "fit", terms and representatives per topic (defaults 8 and 5), passed to plot().

... — Passed to the underlying plotting calls.

Value

x, invisibly. Called for the plot.

Conditions

An error of class sbert_bad_input when type = "fit" and n_topics is missing for a comparison of several candidates.

See Also

compare_topics(); summary().

Examples

```
corpus <- head(dedupe(feedback_translations, column = "translation"), 600)
comparison <- compare_topics(corpus, n_topics = c(4, 6, 8),
                             column = "translation",
                             embeddings = feedback_embeddings)
plot(comparison)
```

---

pool *Pool token embeddings into sentence embeddings*

Description

pool() reduces the token embeddings of each text to one sentence embedding, by mean or CLS pooling. A transformer encoder returns one vector per word piece of a text, whereas a sentence embedding is one vector per text, and the pooling rule is part of the model's definition: Sentence-BERT models average the token vectors over the positions that hold real tokens (mean pooling; Reimers and Gurevych, 2019), while the BGE and mxbai families take the vector of the first, classification token (CLS pooling).

It applies either rule to an array of token embeddings and its attention mask, and by default scales each pooled vector to unit Euclidean (L2) length, the form in which cosine similarity is a dot product. It returns a matrix with one row per text. It is the step encode() runs after the ONNX model, exposed for checking the arithmetic or for pooling token embeddings computed elsewhere.

To embed text, use encode(), which tokenizes, runs the model, and pools with the rule pinned for that model; models(detail = TRUE) lists each model's rule.

Usage

```
pool(
  token_embeddings,
  attention_mask,
  normalize = TRUE,
  method = c("mean", "cls")
)
```

Arguments

`token_embeddings`	Numeric array of token embeddings, shaped texts x tokens x hidden dimensions.
`attention_mask`	Numeric matrix shaped texts x tokens: 1 for a real token, 0 for padding.
`normalize`	Scale each pooled vector to unit L2 length (default `TRUE`).
`method`	`"mean"` (default), the average over positions with mask 1, or `"cls"`, the first token.

Value

A numeric matrix with one row per text and one column per hidden dimension, without dimnames.

References

Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence embeddings using Siamese BERT-networks. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 3982-3992. doi:10.18653/v1/D191410

See Also

`encode()` for embeddings of text; `models()` for the pooling rule of each pinned model.

Examples

```
# Two texts, three token positions, two hidden dimensions; the first text
# has two real tokens, the second one.
hidden <- array(1:12, dim = c(2, 3, 2))
mask <- matrix(c(1, 1, 0, 1, 0, 0), nrow = 2, byrow = TRUE)
pool(hidden, mask)
pool(hidden, mask, method = "cls")
pool(hidden, mask, normalize = FALSE)
```

`predict.sbert_topic_model` *Assign new documents to fitted topics*

Description

`predict()` assigns new documents to the topics of a fitted model without fitting again: it embeds the new texts (or takes their embeddings) and assigns each to the topic whose center has the highest cosine similarity with it, after scaling the centers to unit length. A fitted topic model defines its topics by their centers in the embedding space, so new texts (later responses, a held-out sample, another cohort) can be placed in the same topics, which keeps the topics and their labels fixed and makes the counts comparable across samples.

`distance` is one minus that similarity. A model fitted on segments splits the new texts with the same segmentation settings first and assigns each segment. The fitted model is not changed. The result is a data frame with one row per document, or per segment for a segmented model.

This rule differs from the one that formed the topics: `topics()` fits Lloyd's k-means, which assigns each unit to the nearest raw mean of the normalized embeddings by squared Euclidean distance, and those means are shorter than unit length by different amounts. On the units a model was fitted on, the two rules agree for almost every unit but can differ for a few units on topic boundaries, so `predict()` on the training texts need not reproduce the fitted topics exactly; read the fitted assignment with `as.data.frame(fit, what = "documents")`. For graded rather than single assignments use `topic_membership()`; for the topic mixture of long documents, `topic_gamma()`.

Usage

```
## S3 method for class 'sbert_topic_model'
predict(object, text, model = NULL, embeddings = NULL, batch_size = 32L, ...)
```

Arguments

`object`	A fitted model from `topics()`.
`text`	Character vector of new documents.
`model`	A model from `load_model()`, a pinned model name, or `NULL` (default) for the default model, used to embed `text`; ignored when `embeddings` are given. Its dimension must match the fitted model.
`embeddings`	Numeric matrix with one row per document (one per segment for a segmented model), or `NULL` (default) to encode `text`.
`batch_size`	Texts per model call when encoding (default 32).
`...`	Unused.

Value

A base data frame with one row per document and columns `document_id`, `document_name`, `text`, `topic`, `label` (the fitted topic label), and `distance` (cosine distance to the assigned center). For a segmented model, one row per segment, with `segment` (position in the document) after `document_name`.

Conditions

`sbert_dimension_mismatch` when the embeddings have a different number of columns from the fitted model; `sbert_bad_input` when both `model` and `embeddings` are given, when `embeddings` has the wrong number of rows or non-finite values or a zero row, or when segmentation yields no segment; when encoding, the conditions of `encode()`.

See Also

`topics()` for the fit; `topic_membership()` for graded membership; `topic_gamma()` for document mixtures; `representatives()` for the clearest examples of each topic.

Examples

```
corpus <- head(dedupe(feedback_translations, column = "translation"), 600)
topic_model <- topics(corpus, n_topics = 6, column = "translation",
                      embeddings = feedback_embeddings)

# Assign three messages, giving their precomputed embeddings.
new_messages <- head(corpus, 3)
with(new_messages,
     predict(topic_model, translation,
             embeddings = head(feedback_embeddings, 3)))

# A small model on toy two-dimensional embeddings.
text <- c(
  "Cats chase mice", "Dogs chase balls",
  "Stocks and bonds trade", "Markets price shares"
)
embeddings <- rbind(c(1, 0), c(0.9, 0.1), c(0, 1), c(0.1, 0.9))
toy_model <- topics(text, 2, embeddings = embeddings)
predict(toy_model, "Bulls and bears move markets",
        embeddings = rbind(c(0.2, 0.8)))


if (interactive()) {
  predict(topic_model, c("The pictures were too small.",
                         "I liked the counting game."))
}
```

---

print.sbert_model *Print a loaded model*

Description

`print()` shows a loaded model: the repository id, the pinned revision, the number of embedding dimensions, and the backend. A loaded model is identified by its Hugging Face repository and the revision it is pinned to; together with the embedding dimension and the execution backend, these are what a report needs to state which model produced a set of embeddings.

For the full list of settings of every pinned model, including pooling and prefix, use `models(detail = TRUE)`.

Usage

```
## S3 method for class 'sbert_model'
print(x, ...)
```

Arguments

`x`	An `sbert_model` from `load_model()` or `load_custom()`.
`...`	Ignored.

Value

`x`, invisibly.

See Also

`load_model()`; `models()` for the settings of all pinned models.

Examples

```
# Repository ids and revisions of the pinned models, without loading one.
models(detail = TRUE)


if (interactive()) {
  print(load_model())
}
```

---

print.sbert_topic_corpus

Print a prepared topic corpus

Description

`print()` shows what a corpus prepared by `topic_corpus()` holds: the number of documents and, for a segmented corpus, of segments and their level; the embedding dimension; the embedding model; and the minimum token length and whether tokens are stemmed.

Usage

```
## S3 method for class 'sbert_topic_corpus'
print(x, ...)
```

Arguments

x	A corpus prepared by topic_corpus().
...	Ignored.

Value

x, invisibly.

See Also

topic_corpus(); as.data.frame() for the units.

Examples

```
corpus <- head(dedupe(feedback_translations, column = "translation"), 600)
print(topic_corpus(corpus, column = "translation",
                   embeddings = feedback_embeddings))
```

---

print.sbert_topic_hierarchy

Print a topic hierarchy

Description

print() shows a topic hierarchy: the number of topics and merges, and the merge table: one row per merge, with the cosine distance at which it happens and the two branches it joins.

Usage

```
## S3 method for class 'sbert_topic_hierarchy'
print(x, ...)
```

Arguments

x	A hierarchy built by topic_hierarchy().
...	Ignored.

Value

x, invisibly.

See Also

topic_hierarchy(); as.data.frame(); summary().

Examples

```
text <- c(
  "Cats chase mice", "Dogs chase balls", "Kittens nap in sunshine",
  "Stocks and bonds trade", "Markets price shares", "Banks report profit"
)
embeddings <- rbind(
  c(1, 0), c(0.95, 0.05), c(0.9, 0.1),
  c(0, 1), c(0.05, 0.95), c(0.1, 0.9)
)
tree <- topic_hierarchy(topics(text, n_topics = 3, embeddings = embeddings,
                               n_terms = 3))
print(tree)
```

---

print.sbert_topic_model

Print a topic model

Description

`print()` shows a fitted topic model at a glance: the number of documents (and of segments, for a segmented fit), the number of topics, the embedding model, the clustering algorithm, the size of every topic, and the share of the embedding variance that lies between topics.

Usage

```
## S3 method for class 'sbert_topic_model'
print(x, ...)
```

Arguments

`x`	A topic model fitted by `topics()`.
`...`	Ignored.

Value

`x`, invisibly.

See Also

summary() for coherence and topic diversity; `topic_sizes()` for the labeled size table; as.data.frame() for the tables.

Examples

```
corpus <- head(dedupe(feedback_translations, column = "translation"), 600)
print(topics(corpus, n_topics = 6, column = "translation",
             embeddings = feedback_embeddings))
```

print.sbert_topic_sweep

Print a topic-count comparison

Description

`print()` shows a topic-count comparison: the coherence measure, one row per candidate with its measures, and, when the fitted models are kept, the fitted() call that returns the most coherent candidate.

Usage

```
## S3 method for class 'sbert_topic_sweep'
print(x, ...)
```

Arguments

`x`	A comparison returned by `compare_topics()`.
`...`	Ignored.

Value

`x`, invisibly.

See Also

`compare_topics()`; summary(); as.data.frame().

Examples

```
corpus <- head(dedupe(feedback_translations, column = "translation"), 600)
print(compare_topics(corpus, n_topics = c(4, 6), column = "translation",
                     embeddings = feedback_embeddings))
```

print.summary.sbert_topic_hierarchy

Print a topic hierarchy summary

Description

`print()` shows the nearest neighbor of every topic, as computed by summary(), as a table.

Usage

```
## S3 method for class 'summary.sbert_topic_hierarchy'
print(x, digits = 3L, ...)
```

Arguments

x	A summary returned by summary().
digits	Significant digits for the distances (default 3).
...	Ignored.

Value

x, invisibly.

See Also

summary.sbert_topic_hierarchy().

Examples

```
text <- c(
  "Cats chase mice", "Dogs chase balls", "Kittens nap in sunshine",
  "Stocks and bonds trade", "Markets price shares", "Banks report profit"
)
embeddings <- rbind(
  c(1, 0), c(0.95, 0.05), c(0.9, 0.1),
  c(0, 1), c(0.05, 0.95), c(0.1, 0.9)
)
tree <- topic_hierarchy(topics(text, n_topics = 3, embeddings = embeddings,
                               n_terms = 3))
print(summary(tree), digits = 2)
```

---

print.summary.sbert_topic_model

Print a topic model summary

Description

print() shows the overview of a topic model summary (documents, segments, topics, model, between-topic share of the variance, mean coherence, topic diversity, and a note when k-means did not converge) followed by the table of topics.

Usage

```
## S3 method for class 'summary.sbert_topic_model'
print(x, digits = 4L, ...)
```

Arguments

x	A summary returned by summary().
digits	Significant digits in the table of topics (default 4).
...	Ignored.

Value

x, invisibly.

See Also

summary.sbert_topic_model().

Examples

```
corpus <- head(dedupe(feedback_translations, column = "translation"), 600)
topic_model <- topics(corpus, n_topics = 6, column = "translation",
                      embeddings = feedback_embeddings)
print(summary(topic_model), digits = 2)
```

---

print.summary.sbert_topic_sweep

Print a topic-count comparison summary

Description

print() shows the coherence measure and weighting of a comparison summary made by summary(), followed by its table of one row per segment level.

Usage

```
## S3 method for class 'summary.sbert_topic_sweep'
print(x, digits = 3L, ...)
```

Arguments

x	A summary returned by summary().
digits	Significant digits for the measures (default 3).
...	Ignored.

Value

x, invisibly.

See Also

summary.sbert_topic_sweep().

Examples

```
corpus <- head(dedupe(feedback_translations, column = "translation"), 600)
comparison <- compare_topics(corpus, n_topics = c(4, 6),
                             column = "translation",
                             embeddings = feedback_embeddings)
print(summary(comparison), digits = 2)
```

`reduce_topics` *Merge the topics of a model into fewer topics*

Description

`reduce_topics()` merges the closest topics of a fitted model into `n_topics` topics, giving a coarser model in which every document keeps its place: documents that shared a topic still share one. A model with more topics than the data support has near-duplicate topics that divide one theme between them; merging them differs from fitting a new model with fewer topics, in which documents may be regrouped.

It cuts the hierarchy of `topic_hierarchy()` into `n_topics` groups and merges the topics within each group. The new topic centers are the normalized means of their members' embeddings, and the terms, labels, sizes, representatives, and sums of squares are computed again for the merged topics. The result is a topic model like any other, so every verb applies to it. Names given with `name_topics()` are dropped, because the merged topics need new ones.

Usage

```
reduce_topics(object, n_topics, embeddings = NULL, method = "average")
```

Arguments

`object`	A topic model fitted by `topics()`.
`n_topics`	Number of topics after merging: at least 2 and below the current number.
`embeddings`	The embedding matrix the model was fitted on. Needed only for a model fitted with `keep_embeddings = FALSE`. Default `NULL`.
`method`	Agglomeration method passed to `stats::hclust()` (default `"average"`).

Value

A topic model of class `sbert_topic_model` with `n_topics` topics, structured as a `topics()` fit.

Conditions

Errors of class `sbert_bad_input` when `n_topics` is not below the current number of topics, and `sbert_no_embeddings` when the model holds no embeddings and none are given.

See Also

`topic_hierarchy()` to see which topics would merge; `compare_topics()` to compare fits with different numbers of topics.

Examples

```
corpus <- head(dedupe(feedback_translations, column = "translation"), 600)
topic_model <- topics(corpus, n_topics = 6, column = "translation",
                      embeddings = feedback_embeddings)
reduced <- reduce_topics(topic_model, n_topics = 4)
topic_sizes(reduced, weights = "n")
```

---

representatives *Select representative texts for each topic*

Description

`representatives()` selects representative texts for each topic of a fitted model: it ranks the units of each topic and returns the first `n`. A topic in an embedding model is a region of meaning, and it is read most directly from the texts it contains.

Each unit is assigned to the topic whose unit-length center has the highest cosine similarity with it. The texts closest to a topic's center are typical of it but tend to be long, because long texts resemble the average of many texts; texts that are clearly about one topic and no other are usually more telling. With `rank = "margin"`, the default, units are ranked by their margin: the cosine distance to the second-nearest center minus the distance to their own. With `rank = "distance"` they are ranked by closeness to their own center. Ties go to the shorter text; repeated texts and texts without a word of two or more letters are skipped. By default the units the model was fitted on are ranked, from their stored embeddings; given `text`, any candidate texts are. The result is a data frame with one row per selected text, ordered by topic and rank.

Because the assignment uses cosine similarity to the unit-length centers, while `topics()` fits Lloyd's k-means (squared Euclidean distance to the raw means of the normalized embeddings), a few fitted units on topic boundaries can be ranked under a different topic from the one they were fitted to. The representatives stored at fitting time, shown by `as.data.frame(fit, what = "representatives")`, are the units of each fitted topic closest to its center. For topic words use terms() or `keywords()`.

Usage

```
representatives(
  object,
  text = NULL,
  model = NULL,
  embeddings = NULL,
  n = 3L,
  rank = c("margin", "distance"),
  batch_size = 32L
)
```

Arguments

`object`	A fitted model from `topics()`.

text	Character vector of candidate texts, or NULL (default) for the documents or segments the model was fitted on.
model	A model from load_model(), a pinned model name, or NULL (default) for the default model, used to embed text; ignored when embeddings are given.
embeddings	Numeric matrix with one row per candidate, or NULL (default): the stored embeddings when text is NULL, else encode text.
n	Number of texts per topic (default 3).
rank	"margin" (default) or "distance".
batch_size	Texts per model call when encoding (default 32).

Value

A base data frame with one row per selected text and columns topic, rank, text, distance (cosine distance to the unit's own center), and margin, ordered by topic and rank. When the fitted units are ranked, document_id (and segment, for a segmented model) follow rank. Topics without units have no rows.

Conditions

sbert_no_embeddings when text is NULL and the model stores no embeddings; sbert_dimension_mismatch when the embeddings have a different number of columns from the fitted model; sbert_bad_input when both model and embeddings are given or embeddings has the wrong number of rows; when encoding, the conditions of encode().

See Also

terms() and keywords() for topic words; name_topics() to label topics; plot(fit, type = "representatives") to draw them; topic_membership() for graded membership.

Examples

```
corpus <- head(dedupe(feedback_translations, column = "translation"), 600)
topic_model <- topics(corpus, n_topics = 6, column = "translation",
                      embeddings = feedback_embeddings)
representatives(topic_model, n = 2)
representatives(topic_model, n = 2, rank = "distance")

# Rank new candidate texts, giving their embeddings.
text <- c(
  "Cats chase mice", "Dogs chase balls",
  "Stocks and bonds trade", "Markets price shares"
)
embeddings <- rbind(c(1, 0), c(0.9, 0.1), c(0, 1), c(0.1, 0.9))
toy_model <- topics(text, 2, embeddings = embeddings)
representatives(
  toy_model,
  c("kittens pounce", "bond yields", "pets and prices"),
  embeddings = rbind(c(1, 0.1), c(0.1, 1), c(0.7, 0.7)),
  n = 1
)
```

segment *Split documents into sentences, clauses, or phrases*

Description

`segment()` splits documents into sentences, clauses, or phrases by deterministic rules, with no model involved, so the result is reproducible and needs no download. A long document usually discusses several things, and one embedding of the whole document averages them; shorter units can each be embedded and assigned to a topic on their own, so that a document can be described as a mixture of topics, and each unit stays within the input length of the model.

The `"sentence"` level splits at `.`, `?` and `!`, protecting the periods of abbreviations (see `abbreviations()`), decimal numbers and parenthetical remarks. The `"clause"` level, the default, also splits at `;`, `:`, spaced dashes, and subordinating words such as "which", "where" or "in terms of", while comma-separated lists stay whole. The `"phrase"` level also splits at commas. Quotes, dashes and whitespace are normalized first; letter case is kept. Optional steps merge very short segments into their neighbors, cap segment length at a number of words or model tokens, and drop segments with little alphabetic content, such as citation fragments. The result is a data frame with one row per segment.

The topic verbs segment documents themselves through their `segment` argument (see `topics()` and `topic_corpus()`); call `segment()` directly to inspect the units, or to reuse one segmentation and its embeddings in `topic_gamma()` and `blend()`.

Usage

```
segment(
  text,
  level = c("clause", "sentence", "phrase"),
  merge_below = 0L,
  abbreviations = default_abbreviations(),
  cores = 1L,
  max_tokens = NULL,
  model = NULL,
  min_content = 0
)
```

Arguments

`text`	Character vector of documents. Names are carried into the `document_name` column.
`level`	`"clause"` (default), `"sentence"`, or `"phrase"`.
`merge_below`	Join segments shorter than this many words to a neighbor; `0` (default) keeps every segment as split.
`abbreviations`	Abbreviations, each ending in a period, whose period never ends a sentence (default `abbreviations()`); matched ignoring case.
`cores`	Number of forked processes used to split documents (default 1). Ignored on Windows and for small inputs; the result does not depend on it.

max_tokens	Maximum length of a segment, or NULL (default) for no limit. A longer segment is split again at clause words, ;, :, dashes or commas, or, failing those, before a function word near the limit. Length is counted in words, or in the tokens of model when one is given; a tokenizer yields more tokens than words, so a word limit should sit below the model's token limit.
model	NULL (default), or a model from load_model() whose tokenizer counts the length for max_tokens. Requires max_tokens; splitting then runs in one process.
min_content	Minimum share of alphabetic content, between 0 and 1, for a segment to be kept (see content_ratio()); 0 (default) keeps every segment.

Value

A base data frame with one row per segment and columns document_id (position of the document in text), document_name (its name, or ""), segment (position within the document), and text. Blank documents contribute no rows.

Conditions

sbert_bad_input when model is given without max_tokens.

See Also

abbreviations() for the protected abbreviations; content_ratio() for min_content; topics() and topic_corpus(), which segment through their segment argument; topic_gamma() and blend(), which take this data frame.

Examples

```
feedback <- c(
  first = paste("The task was clear. I liked the colors, the sounds",
                "and the pictures, which were funny."),
  second = "Too hard; I did not understand the question."
)
segment(feedback)
segment(feedback, level = "sentence")
segment(feedback, level = "phrase")

# Abbreviation and decimal periods do not end a sentence.
segment("See Fig. 3 for details. The mean was 2.5 points.",
        level = "sentence")
```

---

stop_words *Get and adjust the stop-word list*

Description

`stop_words()` returns the built-in English list of stop words, extended with `add` or reduced with `remove`. Stop words are frequent function words, such as articles, pronouns, and prepositions, that occur in texts of every topic and so say little about what distinguishes one topic from another; excluding them lets the terms that describe a topic be its content words.

Adding words that every text of a corpus shares, such as the name of the course or the disease a corpus is about, removes them from the topic terms as well. The list filters only the terms and keyword candidates; the text that is embedded is unchanged.

The result is passed as `stop_words =` to `topics()`, `compare_topics()`, `topic_corpus()`, terms(), or `keywords()`. A character vector of one's own replaces the list, and `character()` turns the filter off.

Usage

```
stop_words(language = "en", add = NULL, remove = NULL)
```

Arguments

`language`	Language of the list. Only `"en"` (default) is available.
`add`	Words to add to the list, matched case-insensitively. Default `NULL`.
`remove`	Words to remove from the list. Default `NULL`.

Value

A character vector of lower-case stop words, sorted.

Conditions

An error of class `sbert_bad_input` when `language` is not `"en"`.

See Also

`topics()` and terms(), which use the list; `clean_corpus()` to clean the text itself.

Examples

```
head(stop_words())

# Keep the defaults, add two domain words, and drop one entry
custom <- stop_words(add = c("try", "again"), remove = "against")

corpus <- head(dedupe(feedback_translations, column = "translation"), 600)
topic_model <- topics(corpus, n_topics = 6, column = "translation",
                      embeddings = feedback_embeddings, stop_words = custom)
terms(topic_model, n = 3)
```

`strip_list_markers` *Remove list and enumeration markers from text*

Description

`strip_list_markers()` removes the markers of numbered and lettered lists, such as `"1."`, `"(ii)"`, `"(a)"` and `"IV."`, from the text itself, before segmentation and encoding. Such markers, common in texts taken from documents, forms, or reports, carry no meaning, but they split sentences where there is no sentence boundary, shift the embeddings, and appear in the representative texts.

It removes a one- or two-digit number with a period where a list can be (at the start of the text or of a sentence, or continuing a list); a parenthesized number, single letter, or Roman numeral; and an upper-case Roman numeral with a period. A marker must stand alone, so a number that ends a sentence (`"in the top 20."`), years, decimals, and words in parentheses are kept. The result is `text` with the markers removed and white space collapsed.

The `numbers`, `roman_numerals`, and `section_numbers` arguments of `topics()` filter the terms only. `clean_corpus()` applies this function together with other cleaning steps.

Usage

```
strip_list_markers(text)
```

Arguments

`text`	A character vector without missing values.

Value

`text` with the markers removed, runs of white space collapsed to one space, and each element trimmed.

See Also

`clean_corpus()`, which includes this step; `dedupe()`; `segment()`.

Examples

```
strip_list_markers(c("1. background 2. methods", "(i) first (ii) second"))
strip_list_markers("the concept of (civil) society in (2020)")
strip_list_markers("The student ranks in the top 20. Good work.")
```

summary.sbert_topic_hierarchy *Summarize a topic hierarchy*

Description

summary() lists every topic of a hierarchy with its nearest neighbor among the other topics, the topic whose center has the smallest cosine distance to its own, and the step and height at which the topic first merges in the hierarchy. A topic at a small distance from its neighbor is a near-duplicate of it; a topic that merges only late is distinct from all others.

Usage

```
## S3 method for class 'sbert_topic_hierarchy'
summary(object, ...)
```

Arguments

object	A hierarchy built by topic_hierarchy().
...	Ignored.

Value

A data frame of class summary.sbert_topic_hierarchy with one row per topic and columns topic, label, nearest_topic, nearest_label, distance (cosine distance between the two topic centers), merge_step, and merge_height (the first merge that includes the topic). It prints as a table.

Conditions

An error of class sbert_bad_input for a hierarchy built by an sbert version that did not store the topic distances; rebuild it with topic_hierarchy().

See Also

topic_hierarchy(), reduce_topics(), as.data.frame().

Examples

```
corpus <- head(dedupe(feedback_translations, column = "translation"), 600)
topic_model <- topics(corpus, n_topics = 6, column = "translation",
                      embeddings = feedback_embeddings, label = "terms")
summary(topic_hierarchy(topic_model))
```

`summary.sbert_topic_model` *Summarize a topic model*

Description

`summary()` reports the quality of a fitted topic model: the number of documents (and segments), topics, and the embedding model; the share of the embedding variance between topics; the mean coherence of the topics' terms; and the diversity of the terms. Below this overview, a table gives the size and coherence of every topic, so that small or incoherent topics stand out. A fit whose k-means did not converge is reported as such.

The result is an object that prints as the report; calling `summary()` does not print by itself.

Usage

```
## S3 method for class 'sbert_topic_model'
summary(object, measure = c("npmi", "umass"), n_terms = 10L, ...)
```

Arguments

`object`	A topic model fitted by `topics()`.
`measure`	Coherence measure passed to `coherence()`: `"npmi"` (default) or `"umass"`.
`n_terms`	Top terms per topic used for coherence and diversity (default 10).
`...`	Ignored.

Value

A data frame of class `summary.sbert_topic_model` with one row per topic and columns `topic`, `label`, `n_documents`, `proportion`, and `coherence`. The overview is kept with it and printed above the table; `as.data.frame()` returns the table as a plain data frame.

Conditions

An error of class `sbert_no_terms` when the model has no terms.

See Also

`coherence()` and `topic_diversity()` for the measures; `topic_sizes()`; `print.summary.sbert_topic_model()`.

Examples

```
corpus <- head(dedupe(feedback_translations, column = "translation"), 600)
topic_model <- topics(corpus, n_topics = 6, column = "translation",
                      embeddings = feedback_embeddings)
summary(topic_model)
as.data.frame(summary(topic_model))
```

`summary.sbert_topic_sweep` *Summarize a topic-count comparison*

Description

`summary()` reduces a comparison of candidate numbers of topics to one row per segment level: the range of numbers compared, the number with the highest mean coherence, that candidate's diversity and between-topic variance, and whether every candidate converged. The full table of candidates is given by as.data.frame().

Coherence is one guide among several; the choice of a number of topics should also rest on reading the representative texts of the candidates.

Usage

```
## S3 method for class 'sbert_topic_sweep'
summary(object, ...)
```

Arguments

`object`	A comparison returned by `compare_topics()`.
`...`	Ignored.

Value

A data frame of class `summary.sbert_topic_sweep` with one row per segment level and columns `segment`, `n_candidates`, `min_topics`, `max_topics`, `best_n_topics` (highest coherence; `NA` when no candidate can be scored), `coherence`, `topic_diversity`, `explained`, and `all_converged`. It prints as a table.

See Also

`compare_topics()`, plot(), fitted().

Examples

```
corpus <- head(dedupe(feedback_translations, column = "translation"), 600)
comparison <- compare_topics(corpus, n_topics = c(4, 6, 8),
                             column = "translation",
                             embeddings = feedback_embeddings)
summary(comparison)
```

terms.sbert_topic_model

Compute the terms of each topic

Description

`terms()` computes the ranked terms of each topic of a fitted model: the words that distinguish its texts from the texts of the other topics. Class-based TF-IDF (Grootendorst 2022) treats all the texts of a topic as one document and weights each term by its frequency in the topic against its frequency across all topics, so that a word frequent everywhere ranks low and a word frequent in one topic ranks high.

The terms describe a topic in words and complement its representative texts. They depend only on the topic assignments and the text, so `terms()` recomputes them under any term settings: changing the weighting, the stop words, the number filters, or stemming does not refit the model or encode anything. Every argument left `NULL` keeps the value the model was fitted with. Besides the weight (`score`), each term has its frequency in the topic and its probability within the topic (`beta`), and `sort_by = "beta"` ranks the words a topic uses most rather than the most distinctive ones. The result is a data frame with one row per topic and term.

Usage

```
## S3 method for class 'sbert_topic_model'
terms(
  x,
  n = NULL,
  stop_words = NULL,
  min_term_frequency = NULL,
  min_token_length = NULL,
  weighting = NULL,
  reduce_frequent_words = NULL,
  stem = NULL,
  numbers = NULL,
  roman_numerals = NULL,
  section_numbers = NULL,
  smoothing = 0,
  sort_by = c("score", "beta"),
  ...
)
```

Arguments

`x`	A topic model fitted by `topics()`.
`n`	Terms per topic; `NULL` (default) uses the model's `n_terms`.
`stop_words`	Words excluded from the terms; `NULL` (default) uses the model's list. See `stop_words()`.
`min_term_frequency`	Minimum corpus-wide frequency of a term.

`min_token_length`
: Minimum length of a term, in characters.

`weighting`
: Term weighting: `"ctfidf"` or `"bm25"`.

`reduce_frequent_words`
: Whether to take the square root of within-topic frequencies before weighting.

`stem`
: Whether to merge inflected forms under one Porter stem (needs the SnowballC package).

`numbers`, `roman_numerals`, `section_numbers`
: Whether to `"keep"` or `"remove"` digit-only tokens, Roman-numeral tokens, and section or list numbering.

`smoothing`
: Additive smoothing of `beta` (default 0). With 0, each topic's `beta` over its terms sums to one; a positive value reserves probability for the terms a topic does not use.

`sort_by`
: Order of the terms within a topic: `"score"` (default), the class-based weight, or `"beta"`, the probability within the topic. `n` is applied after ordering.

`...`
: Ignored.

Value

A base data frame with one row per topic and term, and columns `topic`, `label`, `term`, `rank`, `score` (the class-based weight), `frequency` (count within the topic), and `beta` (the probability of the term within the topic).

Conditions

An error of class `sbert_no_terms` when no term passes the filters, and `sbert_missing_package` for `stem = TRUE` without SnowballC.

References

Grootendorst, M. (2022). BERTopic: Neural topic modeling with a class-based TF-IDF procedure. arXiv:2203.05794.

Mendonca, M. and Figueira, A. (2025). Modeling political discourse with Sentence-BERT and BERTopic. arXiv:2510.22904.

See Also

`topics()`; `keywords()` for the keywords of single documents; `coherence()` and `topic_diversity()`, which evaluate the terms; plot() with `type = "terms"`.

Examples

```
corpus <- head(dedupe(feedback_translations, column = "translation"), 600)
topic_model <- topics(corpus, n_topics = 6, column = "translation",
                      embeddings = feedback_embeddings)
terms(topic_model, n = 3)
terms(topic_model, n = 3, weighting = "bm25", sort_by = "beta")
```

topics *Fit semantic topics to documents*

Description

`topics()` fits an embedding-based topic model: it groups texts, or the sentences, clauses, or phrases of texts, by what they mean rather than by the words they share, and each group is a topic. Each unit is represented by a Sentence-BERT embedding (Reimers and Gurevych 2019), a vector in which texts of similar meaning lie close together, and the embeddings are partitioned into groups of semantically similar texts.

Every unit is assigned to one of `n_topics` topics by k-means clustering (Lloyd 1982) of the L2-normalized embeddings, started from farthest-point initial centers. The fit is deterministic: the same input always gives the same topics, and no random seed is involved. Each topic is described by its distinctive terms, scored by class-based TF-IDF (Grootendorst 2022; Mendonca and Figueira 2025), and by its representative units, those nearest its center. By default a topic is labeled by its most central text. Topics are numbered by size, largest first, unless `seeds` fix their order. Unlike latent Dirichlet allocation, the model does not represent a text as a mixture of word distributions, so it does not depend on words co-occurring within texts, which they rarely do in short texts.

The result, an object of class `sbert_topic_model`, shows which kinds of content a corpus holds and how much of the corpus each kind accounts for.

To choose the number of topics and the unit of analysis, compare candidate fits with `compare_topics()`. To fit several models to one corpus, prepare it once with `topic_corpus()`. New texts are assigned to the fitted topics by predict(), and each document's mixture of topics is given by `topic_gamma()`. To merge the topics of an existing fit, use `reduce_topics()`.

Usage

```
topics(
  text,
  n_topics,
  column = NULL,
  model = NULL,
  embeddings = NULL,
  batch_size = 32L,
  iter_max = 100L,
  n_terms = 10L,
  n_representatives = 5L,
  stop_words = default_stop_words(),
  min_term_frequency = 1L,
  min_token_length = 2L,
  weighting = c("ctfidf", "bm25"),
  reduce_frequent_words = FALSE,
  stem = FALSE,
  keep_embeddings = TRUE,
  seeds = NULL,
  seed_embeddings = NULL,
```

```
  fixed_seeds = FALSE,
  cores = 1L,
  numbers = c("keep", "remove"),
  roman_numerals = c("keep", "remove"),
  section_numbers = c("keep", "remove"),
  segment = c("document", "sentence", "clause", "phrase"),
  max_tokens = NULL,
  merge_below = 0L,
  min_content = 0,
  label = c("representative", "terms")
)
```

Arguments

`text`	Documents: a character vector with one document per element, a data frame together with `column`, or a corpus prepared by `topic_corpus()`.
`n_topics`	Number of topics: an integer of at least 2 and at most the number of modeled units.
`column`	Name of the text column when `text` is a data frame. The other columns are carried into the document table. Rows whose text is missing, blank, or the placeholder `"[No abstract available]"` are dropped, and so are their rows of `embeddings`. `NULL` (default) for a character vector.
`model`	The encoder: a model from `load_model()`, a pinned model name from `models()`, or `NULL` (default) for the default model. Not used when `embeddings` are given.
`embeddings`	Optional numeric matrix of precomputed embeddings, one row per modeled unit (per document, or per segment with `segment`). When it is given, no model is loaded. Default `NULL`.
`batch_size`	Texts encoded per model call (default 32).
`iter_max`	Maximum number of k-means iterations (default 100).
`n_terms`	Terms stored per topic (default 10).
`n_representatives`	Representative units stored per topic (default 5).
`stop_words`	Words excluded from the topic terms; default `stop_words()`. `character()` turns the filter off.
`min_term_frequency`	Minimum corpus-wide frequency of a term (default 1).
`min_token_length`	Minimum length of a term, in characters (default 2).
`weighting`	Term weighting: `"ctfidf"` (default) or `"bm25"`, which down-weights terms shared by many topics more strongly.
`reduce_frequent_words`	Whether to take the square root of within-topic term frequencies before weighting (default `FALSE`).
`stem`	Whether to merge inflected forms of a term under one Porter stem, shown as its most frequent form (default `FALSE`; needs the SnowballC package).

keep_embeddings
: Whether to store the normalized embeddings in the model (default TRUE). Without them, reduce_topics(), topic_membership(), and representatives() need the embeddings supplied again.

seeds
: Optional guided topics: a character vector, or a list of character vectors, of seed words or descriptions, one element per seeded topic. The embedding of seed i is the starting center of topic i, and the names of seeds, if any, become topic labels. Default NULL.

seed_embeddings
: Matrix of seed embeddings, one row per seed. Needed when seeds are combined with precomputed embeddings.

fixed_seeds
: If TRUE, the seed centers are held fixed and each unit joins its nearest seed (zero-shot classification); n_topics must equal the number of seeds, and a topic may be empty. Default FALSE.

cores
: Number of forked worker processes for tokenization (default 1; Unix-alikes only). Results do not depend on it.

numbers, roman_numerals, section_numbers
: Whether to "keep" (default) or "remove" digit-only tokens, Roman-numeral tokens, and section or list numbering (1.2.3, 1.) from the topic terms. They change the terms only, not the text that is embedded; to clean the text, use clean_corpus().

segment
: Unit of analysis: "document" (default), or "sentence", "clause", or "phrase", which split each document with segment() and fit the topics to the segments, so that a document can take part in several topics.

max_tokens, merge_below, min_content
: Segmentation options passed to segment(), for segmented fits only: the maximum segment length (in the model's tokens, or in words with precomputed embeddings; default NULL, no limit), the number of words below which a segment is joined to its neighbor (default 0), and the minimum share of letters for a segment to be kept (default 0; see content_ratio()).

label
: How topics are labeled: "representative" (default), the most central text shortened to 60 characters, or "terms", the three most distinctive terms. Named seeds override both.

Value

An object of class sbert_topic_model, a list with

documents One row per modeled unit (document, or segment): document_id, document_name, segment (segmented fits only), text, topic, label, and distance (cosine distance to the topic center), followed by the columns carried from a data frame.

topics One row per topic: topic, label, n_documents, proportion, and withinss.

terms, representatives The stored terms and representative units of each topic.

centers Numeric matrix of topic centers, one row per topic: the k-means means of the L2-normalized embeddings. They are not rescaled, so their norms are below one; verbs that use cosine similarity normalize them first.

`embeddings` The normalized embeddings, or `NULL` when `keep_embeddings = FALSE`.

`diagnostics` Sums of squares (`totss`, `tot_withinss`, `betweenss`), `iterations`, `converged` (whether k-means converged within `iter_max`), `warnings` (warnings raised by `stats::kmeans()`), and `algorithm`.

`model`, `settings` The identity of the embedding model and the settings of the fit, which later verbs reuse.

Read it through its accessors rather than its fields: as.data.frame() with `what = "documents"`, `"topics"`, `"terms"`, or `"representatives"`; `topic_sizes()`; terms(); `representatives()`; and summary(), which also reports a fit that did not converge.

Conditions

Errors of class `sbert_bad_input` for conflicting arguments, embeddings without one row per unit, more topics than distinct embeddings, or a `model`, `embeddings`, or different `segment` combined with a prepared corpus; `sbert_missing_package` for `stem = TRUE` without SnowballC; `sbert_no_terms` when no term passes the filters; and `sbert_clustering_failed` when k-means fails or leaves a topic empty. Arguments of the wrong type or range fail a plain `stopifnot()` check. When a model is used, the errors of `load_model()` and `encode()` apply. A warning of class `sbert_no_converge` when k-means reaches `iter_max` without converging; the fit is still returned. Errors inherit `sbert_error` and warnings `sbert_warning`.

References

Reimers, N. and Gurevych, I. (2019). Sentence-BERT: Sentence embeddings using Siamese BERT-networks. *Proceedings of EMNLP-IJCNLP 2019*, 3982–3992. doi:10.18653/v1/D191410

Grootendorst, M. (2022). BERTopic: Neural topic modeling with a class-based TF-IDF procedure. arXiv:2203.05794.

Mendonca, M. and Figueira, A. (2025). Modeling political discourse with Sentence-BERT and BERTopic. arXiv:2510.22904.

Lloyd, S. P. (1982). Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2), 129–137. doi:10.1109/TIT.1982.1056489

See Also

`compare_topics()` to choose the number of topics; `topic_corpus()` to prepare a corpus once; `name_topics()` to name the topics; plot() for figures; predict() and `topic_gamma()` for new texts and topic mixtures; feedback_embeddings for the example data.

Examples

```
# 600 distinct feedback messages with their precomputed embeddings
corpus <- head(dedupe(feedback_translations, column = "translation"), 600)
topic_model <- topics(corpus, n_topics = 6, column = "translation",
                      embeddings = feedback_embeddings)
topic_model
topic_sizes(topic_model, weights = "n")
terms(topic_model, n = 3)
```

```
# Sentence-level fit: one embedding per sentence
documents <- c(
  "Cats chase mice. Stocks and bonds trade.",
  "Dogs chase balls. Markets price shares."
)
sentence_model <- topics(
  documents, n_topics = 2, segment = "sentence",
  embeddings = rbind(c(1, 0), c(0, 1), c(0.9, 0.1), c(0.1, 0.9))
)
topic_sizes(sentence_model, by = "document")
topic_gamma(sentence_model)
```

`topic_corpus` *Prepare a corpus once for several topic models*

Description

`topic_corpus()` prepares a corpus once for several topic models: it segments the texts, embeds them, and tokenizes them for term scoring, and returns a prepared corpus that `topics()` and `compare_topics()` accept in place of the texts. Fitting topic models usually means fitting several (different numbers of topics, term settings, or seeds on the same texts), and these three steps do not depend on any of those choices; embedding is by far the most expensive step.

Models fitted to the corpus are identical to models fitted to the raw texts. The corpus fixes the embeddings (from `model` or `embeddings`), the segmentation, and the tokenization settings; the settings of each model, such as `n_topics`, `n_terms`, `weighting`, and `seeds`, are still chosen per call. The result is an object of class `sbert_topic_corpus`.

`compare_topics()` builds a corpus itself, so a corpus is needed only when models are fitted one at a time.

Usage

```
topic_corpus(
  text,
  column = NULL,
  model = NULL,
  embeddings = NULL,
  batch_size = 32L,
  stop_words = default_stop_words(),
  min_token_length = 2L,
  stem = FALSE,
  cores = 1L,
  numbers = c("keep", "remove"),
  roman_numerals = c("keep", "remove"),
  section_numbers = c("keep", "remove"),
  segment = c("document", "sentence", "clause", "phrase"),
  max_tokens = NULL,
  merge_below = 0L,
  min_content = 0
)
```

Arguments

text	Documents: a character vector with one document per element, or a data frame together with column.
column	Name of the text column when text is a data frame. The other columns are carried into the document table. Rows whose text is missing, blank, or the placeholder "[No abstract available]" are dropped, and so are their rows of embeddings. NULL (default) for a character vector.
model	The encoder: a model from load_model(), a pinned model name from models(), or NULL (default) for the default model. Not used when embeddings are given.
embeddings	Optional numeric matrix of precomputed embeddings, one row per modeled unit (per document, or per segment with segment). When it is given, no model is loaded. Default NULL.
batch_size	Texts encoded per model call (default 32).
stop_words	Words excluded from the topic terms; default stop_words(). character() turns the filter off.
min_token_length	Minimum length of a term, in characters (default 2).
stem	Whether to merge inflected forms of a term under one Porter stem, shown as its most frequent form (default FALSE; needs the SnowballC package).
cores	Number of forked worker processes for tokenization (default 1; Unix-alikes only). Results do not depend on it.
numbers, roman_numerals, section_numbers	Whether to "keep" (default) or "remove" digit-only tokens, Roman-numeral tokens, and section or list numbering (1.2.3, 1.) from the topic terms. They change the terms only, not the text that is embedded; to clean the text, use clean_corpus().
segment	Unit of analysis: "document" (default), or "sentence", "clause", or "phrase", which split each document with segment() and fit the topics to the segments, so that a document can take part in several topics.
max_tokens, merge_below, min_content	Segmentation options passed to segment(), for segmented fits only: the maximum segment length (in the model's tokens, or in words with precomputed embeddings; default NULL, no limit), the number of words below which a segment is joined to its neighbor (default 0), and the minimum share of letters for a segment to be kept (default 0; see content_ratio()).

Value

An object of class sbert_topic_corpus. Its units (one row per document, or per segment) are given by as.data.frame(); printing it shows the number of documents and segments, the embedding dimension, the model, and the tokenization settings.

Conditions

Errors of class sbert_bad_input when both model and embeddings are given, a document is blank, the embeddings do not have one row per unit, the segmentation produces fewer than two segments,

or segment is passed together with an existing corpus; sbert_missing_package for stem = TRUE without SnowballC. When a model is used, the errors of load_model() and encode() apply.

See Also

topics() and compare_topics(), which take the corpus; as.data.frame() for its units; segment() for the segmentation.

Examples

```
corpus <- head(dedupe(feedback_translations, column = "translation"), 600)
prepared <- topic_corpus(corpus, column = "translation",
                         embeddings = feedback_embeddings)
prepared
topic_sizes(topics(prepared, n_topics = 4))
topic_sizes(topics(prepared, n_topics = 6))
```

---

topic_diversity *Measure the diversity of topic terms*

Description

topic_diversity() measures the diversity of topic terms (Dieng et al. 2020): it pools the top n_terms terms of every topic and returns the proportion of them that are distinct. A value near one means that the topics are described by different words; a low value means that several topics repeat the same words and may be redundant.

It returns a single number between 0 and 1 and complements coherence(), which scores each topic on its own.

Usage

```
topic_diversity(object, n_terms = 10L)
```

Arguments

object	A topic model fitted by topics().
n_terms	Top terms per topic to pool (default 10), capped at the terms a topic has.

Value

A single number between 0 and 1.

Conditions

An error of class sbert_no_terms when the model has no terms.

References

Dieng, A. B., Ruiz, F. J. R., and Blei, D. M. (2020). Topic modeling in embedding spaces. *Transactions of the Association for Computational Linguistics*, 8, 439–453. doi:10.1162/tacl_a_00325

See Also

coherence(); summary(); compare_topics(), which reports both measures per candidate.

Examples

```
corpus <- head(dedupe(feedback_translations, column = "translation"), 600)
topic_model <- topics(corpus, n_topics = 6, column = "translation",
                      embeddings = feedback_embeddings)
topic_diversity(topic_model)
topic_diversity(topic_model, n_terms = 5)
```

---

topic_gamma *Compute document-topic mixtures from segments*

Description

topic_gamma() returns, for each document, its mixture over topics: the share of its segments assigned to each topic, the document-topic distribution conventionally called gamma. A document that discusses two subjects in different sentences belongs partly to two topics, which a single embedding of the whole document cannot express; counting the topics of its segments gives the mixture without any additional parameter.

For a model fitted on segments (segment = in topics()) and text = NULL, it counts the fitted assignments, with no encoding. Otherwise it segments the given documents (with the fitted segmentation, or at level), or takes a segment() data frame as it is, embeds the segments, and assigns each to the topic whose unit-length center has the highest cosine similarity, as predict() does. The result is a data frame with one row per document-topic pair.

The fitted assignments come from Lloyd's k-means in topics(), which uses squared Euclidean distance to the raw means of the normalized embeddings; for a few segments on topic boundaries this can differ from the cosine-similarity rule applied to new text. For graded membership of single units, use topic_membership().

Usage

```
topic_gamma(
  object,
  text = NULL,
  model = NULL,
  embeddings = NULL,
  level = NULL,
  batch_size = 32L,
  cores = 1L,
  dedupe_segments = FALSE,
```

```
  sort_by_length = FALSE,
  top = NULL
)
```

Arguments

object	A fitted model from topics().
text	NULL (default) for the fitted segments of a segmented model; a character vector of documents to segment; or a data frame from segment() (columns document_id and text), used as it is.
model	A model from load_model(), a pinned model name, or NULL (default) for the default model, used to embed the segments; ignored when embeddings are given.
embeddings	Numeric matrix of segment embeddings, one row per segment of text (in the order of segment()), or NULL (default) to encode them.
level	"clause", "sentence", or "phrase" for segmenting a character text, or NULL (default) for the fitted segmentation of a segmented model and "clause" otherwise.
batch_size	Texts per model call when encoding (default 32).
cores	Number of forked processes for segmenting (default 1), passed to segment(); the result does not depend on it.
dedupe_segments	Encode each distinct segment once (default FALSE). Saves time on corpora with many repeated segments; the values can differ from encoding every occurrence by floating-point rounding.
sort_by_length	Batch segments of similar length together when encoding (default FALSE); see encode().
top	NULL (default) for every topic of every document, or a whole number: each document's top largest non-zero shares.

Value

A base data frame with one row per document-topic pair and columns document_id, topic, gamma (the share of the document's segments in the topic), and n_segments. gamma sums to one within each document; documents without segments have no rows. With top, only the largest non-zero shares, ordered by document and share, with a rank column after document_id.

Conditions

sbert_bad_input when text is NULL for a model fitted on whole documents, when model, embeddings or level is given without text, when a data frame lacks document_id or text, when no segment results, or when embeddings has the wrong number of rows; sbert_dimension_mismatch when the embeddings have a different number of columns from the fitted model; when encoding, the conditions of encode().

See Also

segment() for the segments; topic_membership() for graded membership; predict() for single assignments; topic_sizes() for topic totals.

Examples

```
text <- c(
  "Cats chase mice", "Dogs chase balls",
  "Stocks and bonds trade", "Markets price shares"
)
embeddings <- rbind(c(1, 0), c(0.9, 0.1), c(0, 1), c(0.1, 0.9))
fitted <- topics(text, 2, embeddings = embeddings)

# A new document with one sentence on each topic.
mixed <- "Cats chase mice. Stocks and bonds trade."
segment_embeddings <- rbind(c(1, 0), c(0, 1))
topic_gamma(fitted, mixed, embeddings = segment_embeddings)

# Segment once, then reuse the segments and their embeddings.
segments <- segment(mixed, level = "sentence")
topic_gamma(fitted, segments, embeddings = segment_embeddings)

# A model fitted on sentences already holds its segments: no text needed.
sentence_model <- topics(
  c(mixed, "Dogs chase balls. Markets price shares."), 2,
  segment = "sentence",
  embeddings = rbind(c(1, 0), c(0, 1), c(0.9, 0.1), c(0.1, 0.9))
)
topic_gamma(sentence_model)
topic_gamma(sentence_model, top = 1)
```

---

topic_hierarchy *Build the hierarchy of the topics of a model*

Description

topic_hierarchy() clusters the topic centers of a model hierarchically, showing which topics are nearest neighbors, in what order they would merge, and at what distance. The topics of a model are not equally distinct: some lie close together in the embedding space and describe facets of one broader theme, while others stand apart. Topics that merge early, at a small distance, are near-duplicates, which suggests that the number of topics is finer than the data support.

It computes the cosine distances between the topic centers and clusters them with stats::hclust(). The result, an object of class sbert_topic_hierarchy, is deterministic. Use it to choose a smaller number of topics for reduce_topics(), or to report how the topics group into themes.

Usage

```
topic_hierarchy(object, method = "average")
```

Arguments

`object`	A topic model fitted by `topics()`.
`method`	Agglomeration method passed to `stats::hclust()` (default `"average"`).

Value

An object of class `sbert_topic_hierarchy`. as.data.frame() gives the merge table, one row per merge; summary() gives the nearest neighbor of every topic; plot() draws the dendrogram.

Conditions

An error of class `sbert_bad_input` when the model has fewer than two topics.

See Also

`reduce_topics()` to merge topics; `compare_topics()` to compare numbers of topics; `topic_similarity()` for cosine similarities between embeddings.

Examples

```
corpus <- head(dedupe(feedback_translations, column = "translation"), 600)
topic_model <- topics(corpus, n_topics = 6, column = "translation",
                      embeddings = feedback_embeddings, label = "terms")
tree <- topic_hierarchy(topic_model)
tree
summary(tree)
```

`topic_membership` *Compute graded topic membership*

Description

`topic_membership()` computes graded (fuzzy) topic membership: a probability-like weight of every unit on every topic, so that units on a boundary and units squarely inside a topic can be told apart, whereas a hard assignment places each unit in one topic. The weights follow the membership function of fuzzy c-means (Bezdek, 1981): a unit's weight on a topic falls with its distance to that topic's center relative to its distances to the other centers.

The weights are computed from the cosine distance of each unit to each topic center, after scaling the centers to unit length: $u_{ij} \propto d_{ij}^{-2/(m-1)}$, normalized to sum to one over topics, with m the `sharpness`. The topics themselves are not refitted. In high-dimensional embedding spaces distances are similar in size, so the usual $m = 2$ gives nearly uniform weights; the default 1.15 keeps them distinct. The order of topics within a unit does not depend on `sharpness`; the size of the weights does, and should be read relative to the chosen value. The result is a data frame with one row per unit-topic pair, with weights that sum to one within each unit.

The rank-1 topic is the topic predict() assigns, by cosine similarity to the unit-length centers. `topics()` fits Lloyd's k-means, which assigns by squared Euclidean distance to the raw means

of the normalized embeddings; on the fitted units the two rules agree for almost every unit but can differ for a few units on topic boundaries. For the share of a long document's segments in each topic, use `topic_gamma()`.

Usage

```
topic_membership(object, embeddings = NULL, sharpness = 1.15)
```

Arguments

`object`	A fitted model from `topics()`.
`embeddings`	Numeric matrix of embeddings, one row per unit, or `NULL` (default) for the embeddings stored in the model (fitted with `keep_embeddings = TRUE`, the default).
`sharpness`	The fuzzifier m, greater than 1 (default 1.15); values near 1 give crisper weights.

Value

A base data frame with one row per unit-topic pair and columns `document_id` (row of `embeddings`, or of the fitted units), `topic`, `probability`, and `rank` (1 for the unit's highest weight), ordered by unit and rank. Probabilities sum to one within each unit.

Conditions

`sbert_no_embeddings` when `embeddings` is `NULL` and the model stores none; `sbert_dimension_mismatch` when the embeddings have a different number of columns from the fitted model; `sbert_bad_input` when an embedding row is zero.

References

Bezdek, J. C. (1981). *Pattern Recognition with Fuzzy Objective Function Algorithms*. Plenum Press, New York. doi:10.1007/9781475704501

See Also

predict() for single assignments; `topic_gamma()` for document mixtures from segments; `topics()` for the fit.

Examples

```
corpus <- head(dedupe(feedback_translations, column = "translation"), 600)
topic_model <- topics(corpus, n_topics = 6, column = "translation",
                      embeddings = feedback_embeddings)
head(topic_membership(topic_model), 12)

# The textbook fuzzifier spreads the weights more evenly.
head(topic_membership(topic_model, sharpness = 2), 6)
```

topic_palette *Get colors that distinguish topics in plots*

Description

`topic_palette()` returns `n` colors that distinguish topics in plots, from the HCL palette `"Dark 3"`, through `grDevices::hcl.colors(n, palette = "Dark 3")`. Topics are unordered categories, so their colors should differ in hue without suggesting that one topic matters more than another; qualitative palettes from the HCL color space hold lightness and chroma fixed and vary hue only, which serves that purpose.

It is the default of the `colors` argument of `plot.sbert_topic_model()`. Hues alone can be hard to tell apart for readers with color-vision deficiency; the plots therefore also label each topic by number and name. To use other colors, for example the Okabe-Ito palette from `grDevices::palette.colors()`, pass them to `colors`.

Usage

```
topic_palette(n)
```

Arguments

`n`	Number of colors, a positive whole number.

Value

A character vector of `n` hex color codes.

See Also

`plot.sbert_topic_model()`, which uses the palette.

Examples

```
topic_palette(6)

# The Okabe-Ito colors instead, for a toy model of three topics.
text <- c("Cats chase mice", "Dogs chase balls", "Stocks trade",
          "Markets price shares", "Nets learn", "Models train")
embeddings <- rbind(c(1, 0, 0), c(0.9, 0.1, 0), c(0, 1, 0),
                    c(0.1, 0.9, 0), c(0, 0, 1), c(0.05, 0, 0.95))
toy_model <- topics(text, 3, embeddings = embeddings)
plot(toy_model, type = "sizes",
     colors = grDevices::palette.colors(3, palette = "Okabe-Ito"))
```

topic_similarity *Compute cosine similarity between embeddings*

Description

`topic_similarity()` computes the cosine similarity between every row of `x` and every row of `y`, or between all pairs of rows of `x` when `y` is omitted. In a sentence-embedding space, the closeness of two texts in meaning is measured by the cosine of the angle between their embeddings: 1 for the same direction, 0 for unrelated directions, and negative values for opposed ones; cosine similarity ignores vector length, so it compares texts of different lengths on an equal footing.

Rows are scaled to unit length first, so embeddings need not be normalized. The result is a matrix, or with `tidy = TRUE` a long data frame with one row per pair. With `top`, the data frame keeps only the nearest neighbors of each row of `x`: its `top` most similar rows of `y`, most similar first, where a row is not its own neighbor when `y` is omitted.

Any embeddings can be compared: texts from `encode()`, the bundled feedback_embeddings, or rows of another embedding matrix with the same number of columns. To assign texts to fitted topics use predict(); to relate the topics of a model to one another use `topic_hierarchy()`.

Usage

```
topic_similarity(x, y = NULL, tidy = FALSE, top = NULL)
```

Arguments

`x`	Numeric embedding matrix, or a numeric vector (one embedding).
`y`	Numeric embedding matrix or vector, or `NULL` (default) to compare the rows of `x` with each other.
`tidy`	`FALSE` (default) for a matrix; `TRUE` for a data frame.
`top`	`NULL` (default) for every pair, or a whole number: with `tidy = TRUE`, the `top` most similar rows of `y` for each row of `x`.

Value

With `tidy = FALSE`, a numeric matrix of similarities in `[-1, 1]` with one row per row of `x` and one column per row of `y` (or of `x`); a vector counts as one row. With `tidy = TRUE`, a base data frame with one row per pair and columns `from` (row of `x`), `to` (row of `y`), and `similarity`; `from` and `to` are row names when the matrices have them, else row numbers. With `top`, at most `top` rows per row of `x`, ordered by `from` and decreasing similarity (ties by position in `y`), with a `rank` column (1 for the most similar).

Conditions

`sbert_dimension_mismatch` when `x` and `y` have different numbers of columns; `sbert_bad_input` when a row is zero, or when `top` is given without `tidy = TRUE`.

See Also

encode() for embeddings; predict() to assign texts to topics; topic_hierarchy() for similarity between topics.

Examples

```
# Three feedback messages compared with each other.
topic_similarity(head(feedback_embeddings, 3))
topic_similarity(head(feedback_embeddings, 3), tidy = TRUE)

# The three nearest neighbors of each of the 600 messages.
neighbors <- topic_similarity(feedback_embeddings, tidy = TRUE, top = 3)
head(neighbors)

# Toy vectors: rows of x against one vector y.
x <- rbind(c(1, 0), c(0, 1), c(1, 1))
topic_similarity(x, c(1, 1))
```

---

topic_sizes *Compute the size of each topic*

Description

topic_sizes() returns the number and proportion of units in every topic of a fitted model and, with weights, the weighted size and share. The size of a topic can be counted in several ways: counted in distinct texts, it says how varied the content of the topic is; weighted by how often each text occurred, it says how much of the original corpus the topic accounts for.

A topic whose weighted share far exceeds its share of distinct texts is a small set of texts that are repeated many times. For a segmented model, a topic can be counted in segments or, with by = "document", in the documents that have at least one segment in it; a document with segments in several topics then counts in each of them, so the proportions need not sum to one. For a model fitted to documents, the two scales are the same.

Usage

```
topic_sizes(object, weights = NULL, by = c("segment", "document"))
```

Arguments

object	A topic model fitted by topics().
weights	Optional weights: the name of a numeric column carried in the model (for example "n" from dedupe()), or a numeric vector of non-negative weights, one per unit on the chosen scale. Default NULL.
by	Unit counted: "segment" (default), every modeled unit, or "document", the source documents.

Value

A base data frame with one row per topic and columns `topic`, `label`, `n_documents`, and `proportion`, followed by `n_weighted` and `weighted_share` when `weights` are given.

Conditions

An error of class `sbert_bad_input` when `weights` names a column that the model does not carry.

See Also

`dedupe()` for the counts; `topics()`; as.data.frame() for the full tables; plot() with `type = "sizes"`.

Examples

```
corpus <- head(dedupe(feedback_translations, column = "translation"), 600)
topic_model <- topics(corpus, n_topics = 6, column = "translation",
                      embeddings = feedback_embeddings)
topic_sizes(topic_model)
topic_sizes(topic_model, weights = "n")
```

Index

∗ **datasets**
covid, 20
feedback_embeddings, 24
feedback_translations, 25

abbreviations, 5
abbreviations(), *4*, *55, 56*
as.data.frame(), *4*, *47–49*, *59–61*, *67*, *69, 70*, *74*, *79*
as.data.frame.sbert_topic_corpus, 6
as.data.frame.sbert_topic_hierarchy, 7
as.data.frame.sbert_topic_model, 8
as.data.frame.sbert_topic_sweep, 9

blend, 10
blend(), *4*, *55, 56*

cache_dir, 11
cache_dir(), *3, 4*, *12, 13*, *31, 32*, *35, 36*
cache_size, 12
cache_size(), *4*, *12*, *36, 37*
clean_corpus, 13
clean_corpus(), *4*, *19–21*, *57, 58*, *66*, *69*
coherence, 15
coherence(), *4*, *16*, *18*, *60*, *63*, *70, 71*
compare_topics, 16
compare_topics(), *4*, *6*, *9*, *15, 16*, *20*, *22*, *24*, *26*, *39–42*, *49*, *52*, *57*, *61*, *64*, *67, 68*, *70, 71*, *74*
content_ratio, 19
content_ratio(), *4*, *13, 14*, *18*, *56*, *66*, *69*
covid, 20, *25*

dedupe, 21
dedupe(), *4*, *14*, *25*, *58*, *78, 79*

encode, 22
encode(), *4*, *10, 11*, *24*, *29–34*, *42, 43*, *45*, *54*, *67*, *70*, *72*, *77, 78*

feedback_embeddings, *3*, *22*, 24, *25*, *67*, *77*
feedback_translations, *20*, *24*, 25
fitted(), *16–18*, *49*, *61*
fitted.sbert_topic_sweep, 26

install_runtime, 27
install_runtime(), *4*, *31–35*

keywords, 28
keywords(), *4*, *53, 54*, *57*, *63*

load_custom, 29
load_custom(), *4*, *22*, *27*, *32–35*, *46*
load_model, 31
load_model(), *4*, *11*, *17*, *22, 23*, *27–31*, *33–35*, *37*, *44*, *46*, *54*, *56*, *65*, *67*, *69, 70*, *72*

model_download, 34
model_download(), *3, 4*, *12*, *27*, *29*, *32–37*
model_remove, 35
model_remove(), *4*, *12, 13*, *35–37*
model_status, 36
model_status(), *4*, *12, 13*, *33–36*
models, 33
models(), *3, 4*, *12*, *22, 23*, *32, 33*, *35, 36*, *43*, *46*, *65*, *69*

name_topics, 37
name_topics(), *4*, *52*, *54*, *67*

onnxr::onnx_install(), *27*
onnxr::onnx_is_installed(), *27*
onnxr::onnx_model(), *31, 32*

plot(), *4*, *7*, *18*, *37*, *41, 42*, *61*, *63*, *67*, *74*, *79*
plot.sbert_topic_hierarchy, 38
plot.sbert_topic_model, 39
plot.sbert_topic_model(), *76*
plot.sbert_topic_sweep, 41
pool, 42
pool(), *4*, *22*

predict(), *4*, *9, 10*, *22*, *37*, *64*, *67*, *71*, *73–75*, *77, 78*
predict.sbert_topic_model, 44
print.sbert_model, 45
print.sbert_topic_corpus, 46
print.sbert_topic_hierarchy, 47
print.sbert_topic_model, 48
print.sbert_topic_sweep, 49
print.summary.sbert_topic_hierarchy, 49
print.summary.sbert_topic_model, 50
print.summary.sbert_topic_model(), *60*
print.summary.sbert_topic_sweep, 51

reduce_topics, 52
reduce_topics(), *4*, *7*, *18*, *37*, *39*, *59*, *64*, *66*, *73, 74*
representatives, 53
representatives(), *4*, *8–10*, *22*, *37–40*, *45*, *66, 67*

sbert *(*sbert-package*)*, 3
sbert-package, 3
segment, 55
segment(), *4–6*, *10, 11*, *18, 19*, *22, 23*, *58*, *66*, *69–73*
stats::hclust(), *38*, *52*, *73, 74*
stats::kmeans(), *67*
stop_words, 56
stop_words(), *4*, *29*, *62*, *65*, *69*
stopifnot(), *67*
strip_list_markers, 58
strip_list_markers(), *4*, *14*
subset(), *8*
summary(), *4*, *7*, *9*, *15, 16*, *18*, *26*, *37*, *39*, *42*, *47–51*, *67*, *71*, *74*
summary.sbert_topic_hierarchy, 59
summary.sbert_topic_hierarchy(), *50*
summary.sbert_topic_model, 60
summary.sbert_topic_model(), *51*
summary.sbert_topic_sweep, 61
summary.sbert_topic_sweep(), *51*

terms(), *4*, *8, 9*, *28, 29*, *37*, *39, 40*, *53, 54*, *57*, *67*
terms.sbert_topic_model, 62
topic_corpus, 68
topic_corpus(), *4*, *6*, *15*, *17*, *22*, *24*, *46, 47*, *55–57*, *64, 65*, *67*
topic_diversity, 70
topic_diversity(), *4*, *16*, *18*, *60*, *63*
topic_gamma, 71
topic_gamma(), *4*, *22*, *44, 45*, *55, 56*, *64*, *67*, *75*
topic_hierarchy, 73
topic_hierarchy(), *4*, *7*, *18*, *37–39*, *47*, *52*, *59*, *77, 78*
topic_membership, 74
topic_membership(), *4*, *44, 45*, *54*, *66*, *71*, *73*
topic_palette, 76
topic_palette(), *4*, *40*
topic_similarity, 77
topic_similarity(), *4*, *23*, *74*
topic_sizes, 78
topic_sizes(), *4*, *9*, *21*, *37*, *39, 40*, *48*, *60*, *67*, *73*
topics, 64
topics(), *4*, *6*, *8*, *10, 11*, *13*, *15, 16*, *18–20*, *22–24*, *26*, *30*, *37, 38*, *40*, *44, 45*, *48*, *52, 53*, *55–58*, *60*, *62, 63*, *68*, *70–72*, *74, 75*, *78, 79*