Package: sbert (via r-universe)

October 22, 2026

Type Package

Title Topic Modeling with 'Sentence-BERT' Embeddings

Version 0.5.8

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

RoxygenNote 8.0.0

Config/pak/sysreqs libclang-dev

Repository https://sonsoleslp.r-universe.dev

Date/Publication 2026-10-09 17:03:24 UTC

RemoteUrl https://github.com/sonsoleslp/sbert

RemoteRef HEAD

RemoteSha 387023d12ebc6cc73b8a32332fcc60c56ad79af7

Contents

abbreviations . . . 3
as.data.frame.sbert_topic_sweep . . . 3
blend . . . 4
cache_dir . . . 6
cache_size . . . 6
clean_corpus . . . 7
coherence . . . 8
compare_topics . . . 9
content_ratio . . . 12
covid . . . 13
dedupe . . . 14
encode . . . 14
feedback_translations . . . 16
fitted.sbert_topic_sweep . . . 17
install_runtime . . . 18
keywords . . . 18
load_custom . . . 20
load_model . . . 21
model_download . . . 23
model_remove . . . 24
model_status . . . 24
models . . . 25
plot.sbert_topic_hierarchy . . . 25
plot.sbert_topic_model . . . 26
plot.sbert_topic_sweep . . . 28
pool . . . 29
predict.sbert_topic_model . . . 30
print.sbert_model . . . 31
print.sbert_topic_corpus . . . 32
print.sbert_topic_hierarchy . . . 32
print.sbert_topic_model . . . 33
print.sbert_topic_sweep . . . 34
reduce_topics . . . 35
representatives . . . 36
segment . . . 37
stop_words . . . 39

strip_list_markers 40
summary.sbert_topic_model 41
terms.sbert_topic_model 42
topic_corpus 43
topic_diversity 46
topic_gamma 47
topic_hierarchy 49
topic_membership 50
topic_palette 51
topic_similarity 51
topic_sizes 52
topics 53

Index **57**

`abbreviations` *Obtain the Built-in Abbreviation Gazetteer*

Description

Returns the abbreviations whose periods are never treated as sentence boundaries by [segment()]. Matching is case-insensitive and anchored at a word boundary, so ‘"ST."‘ protects ‘"St. Petersburg"‘ but never the end of ‘"cost."‘.

Usage

```
abbreviations()
```

Value

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

Examples

```
head(abbreviations())
```

`as.data.frame.sbert_topic_sweep` *Coerce a Topic-Count Comparison to a Data Frame*

Description

Drops the retained models and the sweep attributes, leaving the plain table of candidates and their measures for further manipulation.

Usage

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

Arguments

x	An ‘sbert_topic_sweep‘ object.
...	Ignored.

Value

A plain data frame without the attached models or sweep attributes.

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)
)
comparison <- compare_topics(text, n_topics = 2:3, embeddings = embeddings)
as.data.frame(comparison)
```

---

blend — *Blend Segment Embeddings with Their Document Context*

Description

Computes context-aware embeddings for text segments: each segment vector is decomposed against its parent document embedding, and the blend keeps ‘alpha‘ of the segment’s context-orthogonal residual plus ‘1 - alpha‘ of the document direction, renormalized to unit length:

$$m = normalize(\alpha(u - (u \cdot d)d) + (1 - \alpha)d)$$

with ‘u‘ the unit-normalized segment embedding and ‘d‘ the unit-normalized parent document embedding.

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 integer ‘document_id‘ and character ‘text‘ columns), one row per segment.
- `documents` — The character vector of documents that was segmented; ‘segments$document_id‘ indexes into it. Required unless both embedding matrices are supplied.
- `model` — A loaded sbert model, a pinned model name, or ‘NULL‘ for the session default (see [load_model()]). Ignored when embeddings are supplied.
- `alpha` — Blend weight in ‘[0, 1]‘ for the segment’s context-orthogonal residual. Default ‘0.5‘.
- `embeddings` — Optional precomputed segment embeddings (one row per row of ‘segments‘). Supply together with ‘document_embeddings‘ to skip encoding.
- `document_embeddings` — Optional precomputed document embeddings (one row per document; ‘segments$document_id‘ indexes its rows).
- `batch_size` — Number of texts encoded per model call when encoding is needed. Default ‘32‘.

Details

This is the principled version of "mixing two embeddings": granularities are mixed per unit, never by pooling units of different levels into one training set (which lets clusters form along a length axis — see length-induced embedding collapse, arXiv:2410.24200). A segment that is ambiguous in isolation inherits its document’s context; identical segment texts from different documents get different blended vectors, which keeps repeated boilerplate from collapsing into a single artificial topic.

‘alpha = 0‘ returns the parent document embedding; ‘alpha = 1‘ keeps only what the segment says beyond its document; the default ‘0.5‘ balances the two. A segment exactly collinear with its document falls back to the document vector.

Value

A numeric matrix with one L2-normalized row per segment, suitable for the ‘embeddings‘ argument of [topics()], [representatives()], and [predict.sbert_topic_model()].

Examples

```
segments <- segment("The model works. It is fast.", level = "sentence")
unit_vectors <- matrix(c(1, 0, 0.6, 0.8), nrow = 2, byrow = TRUE)
document_vectors <- matrix(c(0, 1), nrow = 1)
blend(
  segments,
  alpha = 0.5,
  embeddings = unit_vectors,
  document_embeddings = document_vectors
)
```

cache_dir — *Locate the sbert Model Cache*

Description

Locate the sbert Model Cache

Usage

```
cache_dir()
```

Value

A length-one character vector containing the platform-appropriate user cache directory.

Examples

```
cache_dir()
```

cache_size — *Measure the sbert Cache*

Description

Measure the sbert Cache

Usage

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

Arguments

`cache_dir`	Cache root returned by [cache_dir()].

Value

The total number of bytes currently used by regular files beneath the cache root.

Examples

```
cache_size(tempdir())
```

`clean_corpus` *Clean a Text Corpus Before Encoding*

Description

One place for the text-level cleaning that should happen *before* embedding, so the same clean text feeds the encoder, the topic terms, and the representative examples. Junk characters are always repaired; the structured removals are opt-in. Because cleaning happens up front, encode the result and pass those embeddings on — nothing downstream needs to re-clean.

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`	When ‘text‘ is a data frame, the name of the text column to clean. Every other column rides along and is filtered in step.
`list_markers`	Remove enumeration and list markers (‘1.‘, ‘(i)‘, ‘(a)‘); default ‘TRUE‘. See [strip_list_markers()].
`section_numbers`	Remove multi-level indices (‘1.2.3‘) and enumeration markers; default ‘TRUE‘.
`numbers`	Remove stand-alone numeric tokens (years, counts) while keeping alphanumerics such as ‘covid19‘; default ‘FALSE‘.
`roman_numerals`	Remove stand-alone Roman-numeral tokens (‘ii‘, ‘iv‘, up to 100); default ‘FALSE‘. A few numerals that are also words (‘iv‘, ‘vi‘) go too.
`remove`	Optional character vector of custom Perl-compatible regular expressions. Each match is replaced with a space, before the other steps. The extension point for domain-specific noise the generic cleaners leave — for example ‘remove = "OJ No L?\s*\d+"‘ for European Union Official Journal references, or a journal-citation pattern. Matching is case-insensitive.
`min_content`	Minimum alphabetic-content ratio, in ‘[0, 1]‘, for a document to be kept (see [content_ratio()]). Documents below the floor — citation lists, reference footnotes, number tables — are dropped. ‘0‘ (default) keeps every non-empty document.

Details

The operations, in order: repair junk and strip non-content markup (decode HTML entities and tags; remove URLs, DOIs, emails, and bracketed numeric citations such as ‘[12]‘; normalize curly quotes, dashes, and bullets; drop control, zero-width, BOM, and replacement characters; normalize whitespace); strip list and section numbering; optionally remove numeric and Roman-numeral tokens; then drop documents that fall below ‘min_content‘ (see [content_ratio()]) or are left empty. Content-bearing text — real words, author citations, dates, decimals, inline numbers — is left intact. When ‘text‘ is a data frame the surviving rows are returned whole, so metadata stays aligned.

Value

If ‘text‘ is a character vector, the cleaned vector with dropped documents removed. If a data frame, the surviving rows with the ‘column‘ cleaned in place.

See Also

[strip_list_markers()], [content_ratio()], [dedupe()].

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)

# strip a domain-specific reference form the generic cleaner leaves
clean_corpus("as set out in OJ No L 313 it applies.",
             remove = "OJ No L?\\s*\\d+")
```

---

coherence *Score Topic Coherence*

Description

Computes intrinsic topic coherence from the top terms of a fitted topic model and the co-occurrence structure of the input documents. Coherence rewards topics whose top terms tend to appear together in the same documents, and is the standard quantitative measure for embedding-based topic models (Mimno et al. 2011).

Usage

```
coherence(
  object,
  measure = c("npmi", "umass"),
  n_terms = 10L,
  smoothing = 1,
  token_lists = NULL,
  cores = 1L
)
```

Arguments

`object`	An ‘sbert_topic_model‘ returned by [topics()].
`measure`	Either ‘"umass"‘ (Mimno et al. 2011; larger, i.e. closer to zero, is more coherent) or ‘"npmi"‘ (normalized pointwise mutual information, bounded in ‘[-1, 1]‘; larger is more coherent).
`n_terms`	Number of top terms per topic to score. Capped at the number of terms actually available for each topic.
`smoothing`	Additive smoothing for the UMass co-occurrence count.
`token_lists`	Optional precomputed per-document token lists, aligned to ‘object$documents‘, that skip re-tokenizing the corpus (as produced by [topic_corpus()]). Leave ‘NULL‘ to tokenize internally; the score is identical either way.
`cores`	Number of forked worker processes for tokenization when it is not already cached. Default ‘1‘ (serial). Values above one use ‘parallel::mclapply‘ on Unix-alikes and fall back to serial elsewhere; the result is identical regardless of the count.

Value

A data frame with one row per topic and columns ‘topic‘, ‘label‘, ‘measure‘, ‘n_terms‘, and ‘coherence‘. The corpus-level mean is attached as the ‘"mean_coherence"‘ attribute. Topics with fewer than two scorable terms yield ‘NA‘.

References

Mimno, D., Wallach, H. M., Talley, E., Leenders, M., and McCallum, A. (2011). Optimizing semantic coherence in topic models. EMNLP.

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))
topics <- topics(text, 2, embeddings = embeddings, n_terms = 3)
coherence(topics)
```

`compare_topics` *Compare Topic Counts Before Committing to One*

Description

Fits [topics()] once per candidate topic count — deterministically, on one shared set of embeddings — and returns the quality metrics that justify a granularity choice: mean topic coherence, topic diversity, and the share of embedding variance separated between topics. The verb compares; it never chooses for you. Take the count you want out of the comparison with [fitted()].

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,
  ...
)
```

Arguments

`text` A character vector of documents, a data frame together with ‘column‘, or a prepared [topic_corpus()]. Passing a corpus (or letting ‘compare_topics()‘ build one internally) segments, embeds, and tokenizes the documents once and reuses that work across every candidate, instead of re-tokenizing per candidate.

`n_topics` Integer vector of candidate topic counts, each at least 2 and below the number of modeled units. Default ‘c(5, 10, 15, 20, 25, 30)‘.

`column` When ‘text‘ is a data frame, the name of the column holding the documents to model. Every other column is carried into ‘$documents‘, and rows whose text is missing, blank, or a bibliographic placeholder are dropped. Leave ‘NULL‘ for a character vector.

`model` A loaded sbert model, a pinned model name, or ‘NULL‘ for the session default. Ignored when ‘embeddings‘ is supplied.

`embeddings` Optional precomputed embedding matrix, one row per document (or per segment when ‘segment‘ is not ‘"document"‘); supply it to avoid re-encoding.

`measure` Coherence measure, ‘"npmi"‘ (default) or ‘"umass"‘.

`n_terms` Top terms per topic used for coherence and topic_diversity. Default ‘10‘.

`n_representatives` Representative documents kept per topic in each retained model. Default ‘1‘; raise it when the stored models are meant to be inspected rather than only compared.

`keep_models` Whether to retain every fitted model so the chosen granularity needs no refitting. Default ‘TRUE‘; set ‘FALSE‘ to return only the comparison table when memory matters.

`batch_size` Number of texts encoded per model call when encoding is needed. Default ‘32‘.

`cores` Number of forked worker processes. Default ‘1‘ (serial). Above one, the shared corpus is tokenized in parallel and the candidates — which are independent and deterministic — are fitted in parallel, on Unix-alikes (serial fallback on Windows). Results are identical for any core count.

`segment` The unit the candidates are fitted on, as in [topics()]: ‘"document"‘ (default), ‘"sentence"‘, ‘"clause"‘, or ‘"phrase"‘. Give several levels — for example ‘c("document", "sentence", "clause")‘ — to compare the segmentations as well as the counts: every level is fitted across every candidate, the table gains a leading ‘segment‘ column, the plot draws one line per level, and [fitted()] takes ‘segment =‘ alongside ‘n_topics‘. ‘max_tokens‘, ‘merge_below‘, and ‘min_content‘ then apply to the segmented levels only. With several levels, precomputed ‘embeddings‘ are a list of matrices named by level.

`max_tokens` Optional cap on segment length, passed to [segment()]. When a ‘model‘ is used, the budget counts that model’s exact tokens; with precomputed ‘embeddings‘ it counts words. Only with a segmented fit.

`merge_below` Re-join segments shorter than this many words into their neighbor, passed to [segment()]. Only with a segmented fit.

`min_content` Minimum alphabetic-content ratio for a segment to be kept, passed to [segment()]. Only with a segmented fit.

`...` Further arguments passed to [topics()] (for example ‘stop_words‘ or ‘weighting‘).

Details

There is no single correct topic count; this verb replaces the habit of picking one by feel with a table you can defend. Coherence typically falls as counts grow while separation rises, so look for the count after which coherence stops improving (or starts degrading) rather than a global maximum.

Value

A base data frame with one row per candidate and columns ‘n_topics‘, ‘coherence‘ (corpus mean), ‘topic_diversity‘, and ‘explained‘ (between-topic share of total variance), preceded by ‘segment‘ when several levels were compared. The coherence measure is recorded in the ‘measure‘ attribute. When ‘keep_models = TRUE‘ the result also carries class ‘sbert_topic_sweep‘ and the fitted models, which [fitted()] extracts and [plot.sbert_topic_sweep()] draws.

See Also

[fitted()] to pull one fitted model out of the comparison.

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)
)
comparison <- compare_topics(text, n_topics = 2:3, embeddings = embeddings)
comparison
fitted(comparison, n_topics = 2)

# Documents against sentences, in one table and one plot (offline: one
# embedding matrix per level, one row per unit of that 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
fitted(levels, n_topics = 2, segment = "sentence")
```

`content_ratio` *Alphabetic Content Ratio of Text*

Description

The fraction of a string's non-space characters that are letters. A domain-agnostic measure of how much of a unit is prose rather than numbers, punctuation, or symbols: real sentences score near 1, while citation fragments, reference lists, number tables, and identifiers (‘"OJ No L 297, 24.11.1979, p. 1."‘, ‘"doi:10.1007/..."‘) score low. Used by the ‘min_content‘ filters of [segment()] and [topics()].

Usage

```
content_ratio(text)
```

Arguments

`text`	A character vector.

Value

A numeric vector in ‘[0, 1]‘; empty or space-only elements score ‘0‘.

Examples

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

`covid` *COVID-19 Research Abstracts*

Description

Abstracts of peer-reviewed research on COVID-19, drawn from a bibliographic database export and covering its effects on education, children, schools, and society. Longer and more technical than [feedback_translations], with a publication year on every record, the corpus is a realistic benchmark for the package's embedding and topic-modeling workflow — including temporal analysis across the pandemic years.

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]"‘; 3,671 of the abstracts are distinct and the longest is 5,285 characters.

Editorial notices (retractions, corrections, and errata) are removed during preparation, so every row is a research abstract or a placeholder.

Source

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

Examples

```
table(covid$Year)

content <- covid$Abstract[covid$Abstract != "[No abstract available]"]
nchar(content[1])
```

dedupe — *Deduplicate a Text Corpus with Frequencies*

Description

Collapses a character vector to its distinct non-blank texts, keeping the order of first appearance and counting how often each text occurred. Embedding and clustering operate on the distinct texts (so repeated templates cannot drag the cluster geometry), while the returned counts carry the original frequencies back into reporting, for example through [topic_sizes()].

Usage

```
dedupe(text)
```

Arguments

`text`	A character vector. ‘NA‘ and blank (whitespace-only) elements are dropped before deduplication.

Value

A base data frame with one row per distinct text and columns ‘text‘ (in order of first appearance) and ‘n‘ (number of occurrences).

Examples

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

encode — *Encode Text with Sentence-BERT*

Description

Encode Text with Sentence-BERT

Usage

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

Arguments

`text` Character vector of sentences. Empty strings are supported; missing values are rejected.

`model` A loaded [sbert_model][load_model()], the name of a pinned model from [models()], or ‘NULL‘ for the default ‘all-MiniLM-L6-v2‘. Names are loaded lazily and kept in a session cache; the first use of a not-yet-installed model offers a one-time verified download (interactive prompt, or ‘options(sbert.download = TRUE)‘ in scripts).

`batch_size` Positive number of sentences processed per ONNX call.

`normalize` Whether to return row-wise L2-normalized embeddings.

`prefix` Text prepended to every input before tokenization. ‘NULL‘ (default) uses the model’s pinned prefix (for example ‘"query: "‘ for E5 models, which require one); ‘""‘ disables prefixing.

`sort_by_length` When ‘TRUE‘, group inputs of similar length into the same batch before encoding and restore the original order afterwards. Sequences in a batch are padded to the longest member, so mixed-length input makes the model compute over padding: on clause-level segments 60.1 measured at 1.9x faster end to end. Whole documents of similar length gain little (1.1x). Off by default because regrouping changes batch composition, which moves embeddings by around 1e-7 — enough to flip a rare borderline topic assignment, the same trade-off as ‘dedupe_segments‘ in [topic_gamma()]. Row order of the result is unaffected.

`cache` Optional path to an embedding cache file. When supplied, every document is looked up by a SHA-256 digest of its own text together with the model identity, revision, dimension, maximum length, pooling method and ‘normalize‘ setting; only documents with no matching entry are encoded, and the file is updated with whatever was newly computed. Corpora are usually edited rather than replaced — a few records corrected, a year appended, rows reordered — and encoding is about 97 reuse dominates. Changing 38 documents of 3,847 cost 0.59 s against 57 s to re-encode the corpus. Repeated documents within a single call are also encoded once (295 of the 3,847 bundled ‘covid‘ abstracts are duplicates).

The key deliberately excludes ‘batch_size‘ and ‘sort_by_length‘, which shift embeddings by around 1e-7 through batch composition alone. Cached values are therefore stable to whatever computed them first rather than bit-identical to a fresh encode under different batching — the same trade-off as ‘sort_by_length‘ itself. A cache that cannot be read, or that was written for a different embedding dimension, is discarded with a warning and rebuilt rather than trusted.

Value

A numeric matrix with one row per input and one column per embedding dimension of the loaded model.

Examples

```
# Embeddings need a downloaded model, so the full workflow is in the
# interactive block below. Segmenting needs none.
segment("One sentence. Another sentence.", level = "sentence")
```

```
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 <- segment(covid$Abstract[1:50], level = "clause")
  embeddings <- encode(segments$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 <- encode(covid$Abstract, model, cache = store)
  embeddings <- encode(covid$Abstract, model, cache = store)
}
```

`feedback_translations` *AI-generated Mathematics Feedback with English Translations*

Description

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 is a realistic, quirk-preserving benchmark for the package's embedding, similarity, segmentation, and topic-modeling workflow: translations repeat (templates such as "Try again." recur), some source messages remain in their original language, and a few are identical to their translation.

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. Messages are at most 193 characters long.

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.

Examples

```
head(feedback_translations)

segment(
  head(feedback_translations$translation, 5),
  level = "sentence"
)
```

`fitted.sbert_topic_sweep` *Extract One Fitted Model from a Topic-Count Comparison*

Description

Returns the model [compare_topics()] already fitted for a given topic count, so choosing a granularity from the comparison table costs no refitting.

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 topic count to extract. Must be one of the candidates in the comparison.
`segment`	The segment level to extract when the comparison spans several (‘compare_topics(segment = c(...))‘); one of its levels. Leave ‘NULL‘ for a single-level comparison.
`...`	Unused, present for generic consistency.

Value

The ‘sbert_topic_model‘ fitted for that candidate.

Examples

```
text <- c(
  "Cats chase mice", "Kittens chase mice too", "Cats nap daily",
  "Stocks and bonds trade", "Markets price shares", "Banks report profit"
)
embeddings <- rbind(
  c(1, 0, 0), c(0.98, 0.02, 0), c(0.96, 0, 0.04),
  c(0, 1, 0), c(0.02, 0.98, 0), c(0, 0.96, 0.04)
)
comparison <- compare_topics(
  text, n_topics = 2:3, embeddings = embeddings, n_terms = 3
)
fitted(comparison, n_topics = 2)
```

install_runtime *Install ONNX Runtime*

Description

Explicitly downloads the native ONNX Runtime used by the CRAN package ‘onnxr‘. This is never called during package installation or loading.

Usage

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

Arguments

cuda	Passed to [onnxr::onnx_install()]. Use ‘FALSE‘ for the portable CPU runtime, ‘TRUE‘ to request CUDA on supported systems, or ‘NULL‘ for automatic detection.

Value

The result returned by [onnxr::onnx_install()], invisibly.

Examples

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


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

keywords *Extract Keywords from Documents by Embedding Similarity*

Description

Ranks each document’s own words and phrases by cosine similarity between their embeddings and the document embedding, computed with the same model, and selects the top ‘n‘ by maximal marginal relevance so the keywords are relevant without being redundant (the KeyBERT design).

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`	A character vector of documents. Names, when present, are carried into the ‘document_name‘ column.
`model`	A loaded sbert model, a pinned model name, or ‘NULL‘ for the session default.
`n`	Maximum keywords returned per document. Default ‘10‘.
`ngrams`	Maximum phrase length in tokens. Default ‘2‘ (unigrams and bigrams).
`topic_diversity`	Maximal-marginal-relevance trade-off in ‘[0, 1)‘: ‘0‘ ranks purely by similarity, larger values penalize keywords similar to ones already selected. Default ‘0.3‘.
`stop_words`	Words excluded from candidates. Defaults to [stop_words()].
`min_token_length`	Minimum character length of a candidate token. Default ‘3‘.
`batch_size`	Number of texts encoded per model call. Default ‘32‘.

Details

Candidates are consecutive-token n-grams (lengths ‘1‘ to ‘ngrams‘) drawn from the document after stop-word removal, so a candidate phrase never crosses a removed stop word silently: "analysis of networks" yields the candidates "analysis", "networks", and "analysis networks".

Value

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

Examples

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


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

```
    n = 5
  )
}
```

`load_custom` *Load an Arbitrary Hugging Face Embedding Model*

Description

Loads any public Hugging Face repository that ships an ONNX encoder export and a ‘tokenizer.json‘, returning the same ‘sbert_model‘ object as [load_model()] so that [encode()], [topics()], and every downstream verb work unchanged.

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`	Optional commit hash to pin. ‘NULL‘ resolves the current revision on first download (recorded in the manifest thereafter).
`onnx_path`	Repository path of the ONNX graph. ‘NULL‘ auto-detects ‘onnx/model.onnx‘ or ‘model.onnx‘.
`tokenizer_path`	Repository path of the tokenizer. Default ‘"tokenizer.json"‘.
`pooling`	‘"mean"‘ or ‘"cls"‘. ‘NULL‘ auto-detects from ‘1_Pooling/config.json‘ and fails with a clear message when absent.
`prefix`	Text prepended to every input by [encode()]. ‘NULL‘ (default) keeps the recorded value (‘""‘ on first use); pass ‘""‘ to clear a recorded prefix.
`max_length`	Maximum word pieces per input. ‘NULL‘ auto-detects from ‘sentence_bert_config.json‘, falling back to 512.

cache_dir	Cache root returned by [cache_dir()].
backend	ONNX execution backend accepted by [onnxr::onnx_model()].
threads	Positive number of inference threads.
verify	Whether to verify recorded byte sizes and SHA-256 hashes when loading from an existing cache.
quiet	Whether to suppress download progress.

Details

This is the escape hatch below the curated registry. On first use the current revision is resolved (unless ‘revision‘ is given), the artifacts are downloaded, and their byte sizes and SHA-256 values are recorded in a local manifest ("trust on first use"); later loads verify the files against that manifest and never touch the network. Pooling, prefix, and maximum length are auto-detected from the repository’s Sentence-Transformers configuration when present, and must be supplied explicitly otherwise. The ONNX input signature and embedding dimension are read from the graph itself. Unlike registry models, the package makes no numerical-parity claim: you vouch for the configuration.

Value

An object of class ‘sbert_model‘.

Examples

```
# Cache root used for downloaded and custom models.
cache_dir()


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

---

load_model *Load a Pinned Sentence-BERT Model*

Description

Loads already-downloaded model artifacts. The function never downloads model files or native libraries. See [models()] for the available pinned models.

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 listed by [models()].
`cache_dir`	Cache root returned by [cache_dir()].
`backend`	ONNX execution backend accepted by [onnxr::onnx_model()].
`threads`	Positive number of inference threads, or ‘"auto"‘ to use the machine’s performance-core count. Inference is the dominant cost of [encode()] — about 88 that threads. The default ‘1‘ is deliberate: it keeps runs reproducible across machines and keeps the package within the two-core limit that check environments impose. Raising it is free of numerical consequence — output was ‘identical()‘ at 1, 2, 4, 6 and 8 threads across all 45 measured runs — but expect a modest gain, bounded by *performance* cores rather than logical ones. Over three repetitions at 200, 600 and 1,500 documents on an Apple M4 (4 performance + 6 efficiency), median speedups against a single thread were 1.22x at two threads, 1.29x at four, 1.22x at six and 1.18x at eight. Four threads was fastest at the two larger sizes (1.34x at 1,500 documents); beyond that the efficiency cores add contention rather than throughput. Threaded timings are also noisy — run-to-run spread reached 26 under 3 Set ‘threads = "auto"‘ for the knee of that curve, or an integer to pin it.
`verify`	Whether to verify file sizes and SHA-256 hashes before loading.

Value

An object of class ‘sbert_model‘.

Examples

```
# Pinned models that can be loaded.
models()


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

`model_download` *Download a Pinned Sentence-BERT Model*

Description

Downloads the official ONNX graph and tokenizer of a pinned model only after this function is called. Files are locked to an immutable repository revision and checked against package-controlled SHA-256 values. See [models()] for the available models; the default remains ‘all-MiniLM-L6-v2‘.

Usage

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

Arguments

`model`	Name of a pinned model listed by [models()].
`cache_dir`	Cache root returned by [cache_dir()].
`quiet`	Whether to suppress download progress.
`timeout`	Minimum download timeout in seconds.

Value

Invisibly, the model directory.

Examples

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


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

model_remove — *Remove an Installed Model from the Cache*

Description

Remove an Installed Model from the Cache

Usage

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

Arguments

cache_dir	Cache root returned by [cache_dir()].
model	Name of a pinned model listed by [models()].

Value

Invisibly, whether the model directory no longer exists.

Examples

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

model_status — *Inspect an Installed Model*

Description

Reports whether each required artifact exists and passes its pinned byte-size and SHA-256 checks.

Usage

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

Arguments

cache_dir	Cache root returned by [cache_dir()].
model	Name of a pinned model listed by [models()].

Value

A data frame with one row per required artifact.

Examples

```
model_status(tempdir())
```

models — *List the Available Pinned Sentence-BERT Models*

Description

Every supported model is locked to an immutable Hugging Face revision and verified by byte size and SHA-256 before use. Pass the ‘model‘ column value to any verb that takes a model, for example ‘encode(text, model = "bge-small-en-v1.5")‘.

Usage

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

Arguments

detail	Whether to include the technical columns. The default ‘FALSE‘ returns the menu (‘model‘, ‘dimensions‘, ‘max_tokens‘, ‘languages‘, ‘size_mb‘); ‘TRUE‘ adds ‘pooling‘ (‘"mean"‘ or ‘"cls"‘), ‘prefix‘ (text automatically prepended by [encode()]), ‘engine‘ (‘"onnx"‘ or ‘"static"‘), ‘license‘, ‘id‘, and ‘revision‘.

Value

A data frame with one row per model.

Examples

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

plot.sbert_topic_hierarchy — *Plot a Topic Hierarchy Dendrogram*

Description

Draws the merge tree as a dendrogram labeled with topic labels, with cosine distance on the height axis. Early merges at small heights are near-duplicate topics.

Usage

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

Arguments

x	An ‘sbert_topic_hierarchy‘ object.
main	Plot title.
cex	Character expansion factor for labels.
...	Passed to [plot.dendrogram()].

Value

The hierarchy object, invisibly.

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, 3, embeddings = embeddings, n_terms = 3))
plot(tree)
```

---

plot.sbert_topic_model

Plot a Semantic Topic Model

Description

Draws one of five deterministic base-graphics views of a fitted topic model.

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` An ‘sbert_topic_model‘ returned by [topics()].

`type` One of ‘"sizes"‘ (document count per topic), ‘"terms"‘ (top terms per topic), ‘"representatives"‘ (a ranked text list of the centroid-nearest documents per topic), ‘"fit"‘ (a per-topic report with all three term views – count, TF-IDF, beta – followed by the representative documents, one row per topic), or ‘"map"‘ (a two-dimensional classical-MDS projection of the document embeddings, colored by topic). The ‘"map"‘ view requires a model fitted with ‘keep_embeddings = TRUE‘.

`by` For ‘type = "terms"‘, one or more of ‘"score"‘ (class-based TF-IDF, the default and most distinctive terms), ‘"beta"‘ (the generative within-topic word probability), and ‘"frequency"‘ (the raw within-topic count). A single value draws one panel per topic; several values draw one row per topic with a column for each metric.

`topics` For ‘type‘ ‘"terms"‘, ‘"representatives"‘, or ‘"fit"‘, an optional vector of topic numbers to restrict the plot to. Defaults to all topics.

`per_topic` For ‘type‘ ‘"terms"‘, ‘"representatives"‘, or ‘"fit"‘, draw a separate figure for each topic instead of arranging all topics into one gridded figure. Most useful with a device or ‘knitr‘ chunk that keeps every figure (each topic then becomes its own image).

`n_terms` Number of top terms shown per panel when ‘type‘ is ‘"terms"‘ or ‘"fit"‘.

`n_representatives` Number of representative documents shown per panel when ‘type‘ is ‘"representatives"‘ or ‘"fit"‘. When more are requested than the model stored at fit time, they are recomputed from the retained embeddings; if the model was fitted with ‘keep_embeddings = FALSE‘, the stored set is used and a warning is issued.

`colors` Optional vector of topic colors; defaults to [topic_palette()].

`max_points` Maximum documents drawn when ‘type = "map"‘. Larger corpora are thinned to a deterministic stratified subsample so the classical-MDS projection stays tractable.

`main` Optional figure title for ‘type = "fit"‘. The default names the layout (term columns and representative documents).

`...` Unused; present for S3 compatibility.

Value

Invisibly, ‘x‘.

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

`plot.sbert_topic_sweep`

Plot a Topic-Count Comparison

Description

‘type = "metrics"‘ (the default) draws coherence, topic diversity, and between-topic variance against the candidate topic counts, marking the count with the highest coherence. A comparison over several segment levels draws one line per level in each panel, told apart by color, point shape, and line type, with a legend. There is no single correct topic count; the useful signal is the count after which coherence stops improving, not a global maximum.

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"‘.
`n_topics`	For ‘type = "fit"‘, the candidate count whose models are drawn. Required when the comparison holds several candidates.

segment — For ‘type = "fit"‘, the segment levels to draw; defaults to every level in the comparison.

n_terms, n_representatives — For ‘type = "fit"‘, passed to [plot.sbert_topic_model()].

... — Passed to the underlying plotting calls.

Details

‘type = "fit"‘ draws the per-topic fit report of [plot.sbert_topic_model()] for the retained model at ‘n_topics‘, one figure per segment level, so the same topic count can be read across segmentations side by side.

Value

‘x‘, invisibly.

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)
)
comparison <- compare_topics(text, n_topics = 2:3, embeddings = embeddings)
plot(comparison)
```

---

pool — *Pool Token Embeddings into Sentence Embeddings*

Description

Applies the pooling strategy of the loaded model — attention-mask-aware mean pooling (the Sentence-BERT default) or CLS-token pooling (used by the BGE and mxbai families) — and optional row-wise L2 normalization.

Usage

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

Arguments

token_embeddings	Numeric array shaped batch by sequence by hidden.
attention_mask	Numeric matrix shaped batch by sequence, containing only zero and one.
normalize	Whether to L2-normalize every sentence vector.
method	‘"mean"‘ (default) or ‘"cls"‘ (first token).

Value

A numeric matrix shaped batch by hidden.

Examples

```
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")
```

---

predict.sbert_topic_model

Assign New Documents to Fitted Topics

Description

Predicts the topic of unseen documents by embedding them and assigning each to the nearest stored topic centroid under cosine distance. The fitted model is not modified. Supply either a loaded ‘model‘ (to embed ‘text‘) or a precomputed embedding matrix whose rows align with ‘text‘.

Usage

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

Arguments

object	A fitted [topics()] model.
text	Character vector of new documents. A model fitted with ‘segment = "sentence"‘, ‘"clause"‘, or ‘"phrase"‘ splits them the same way first (with the fitted ‘max_tokens‘, ‘merge_below‘, and ‘min_content‘) and assigns every segment.
model	A loaded [sbert_model][load_model()], a pinned model name, or ‘NULL‘ for the default model; ignored when ‘embeddings‘ are supplied. The embedding dimension must match the fitted model.
embeddings	Optional numeric matrix with one row per document, or one row per segment for a segmented model.
batch_size	Batch size passed to [encode()] when ‘model‘ is used.
...	Unused; included for S3 compatibility.

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 centroid). For a segmented model, one row per segment, with ‘document_id‘ naming the source document and ‘segment‘ its position within it.

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)
predict(fitted, "Bulls and bears move markets", embeddings = rbind(c(0.2, 0.8)))
```

---

print.sbert_model *Print a Loaded Model*

Description

Shows the model’s identity and the settings inference will run under: pinned revision, embedding dimensions, backend, and thread count.

Usage

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

Arguments

x	An ‘sbert_model‘ object.
...	Ignored.

Value

The model object, invisibly.

Examples

```
# Pinned models that can be loaded and printed.
models()


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

print.sbert_topic_corpus

Print a Prepared Topic Corpus

Description

Shows what the corpus holds and can be reused for: document and unit counts, the segmentation level, and whether embeddings are attached.

Usage

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

Arguments

x	An ‘sbert_topic_corpus‘ object.
...	Ignored.

Value

The corpus object, invisibly.

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))
print(topic_corpus(text, embeddings = embeddings))
```

print.sbert_topic_hierarchy

Print a Topic Hierarchy

Description

Shows the merge table: one row per merge, with its height and the two branches that fused.

Usage

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

Arguments

x	An ‘sbert_topic_hierarchy‘ object.
...	Ignored.

Value

The hierarchy object, invisibly.

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, 3, embeddings = embeddings, n_terms = 3))
print(tree)
```

---

print.sbert_topic_model

Print a Semantic Topic Model

Description

Shows the fit at a glance: how many documents and topics, the variance explained, and each topic with its size and its most distinctive terms.

Usage

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

Arguments

x	An ‘sbert_topic_model‘ object.
...	Ignored.

Value

The topic model object, invisibly.

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)
)
print(topics(text, n_topics = 2, embeddings = embeddings, n_terms = 3))
```

---

print.sbert_topic_sweep

Print a Topic-Count Comparison

Description

Shows one row per candidate with its quality measures, and the call that pulls the best-scoring fitted model back out.

Usage

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

Arguments

x	An ‘sbert_topic_sweep‘ object.
...	Ignored.

Value

The sweep object, invisibly.

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)
)
comparison <- compare_topics(text, n_topics = 2:3, embeddings = embeddings)
print(comparison)
```

reduce_topics — *Reduce a Fitted Topic Model to Fewer Topics*

Description

Cuts the topic hierarchy (see [topic_hierarchy()]) at the requested count and rebuilds the model: documents keep their cluster memberships (merged, never re-clustered), centroids are recomputed from the member documents, and terms, labels, sizes, and representatives are derived afresh for the merged topics. The result is a full ‘sbert_topic_model‘ — every downstream verb (‘summary()‘, ‘coherence()‘, ‘predict()‘, [representatives()], plots) works unchanged.

Usage

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

Arguments

object	An ‘sbert_topic_model‘ returned by [topics()].
n_topics	Target number of topics, at least 2 and below the current count.
embeddings	The document embedding matrix used to fit ‘object‘. Only needed when the model was fitted with ‘keep_embeddings = FALSE‘; models fitted with ‘keep_embeddings = TRUE‘ carry their embeddings.
method	Agglomeration method passed to [stats::hclust()]. Default ‘"average"‘.

Value

An ‘sbert_topic_model‘ with ‘n_topics‘ topics.

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)
)
topics <- topics(
  text, 3,
  embeddings = embeddings, n_terms = 3, keep_embeddings = TRUE
)
reduce_topics(topics, 2)
```

representatives *Representative Text Units for Every Topic*

Description

Selects, for each fitted topic, the text units (documents, sentences, or clause segments) that best evidence it. The default ranking is by *distinctiveness margin* — the unit's cosine distance to its second-best centroid minus the distance to its own — rather than by raw closeness. Raw closeness systematically favors long units, because topic centroids are averages of whole-document embeddings and long units resemble whole documents; the margin instead rewards units that are unambiguously about one topic and no other, which are naturally short and pointed. Ties break toward the shorter unit, then alphabetically, so the selection is fully deterministic.

Usage

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

Arguments

object	A fitted [topics()] model.
text	Character vector of candidate units, for example the ‘text‘ column of [segment()]. ‘NULL‘ (the default) ranks the units the model was fitted on — its documents, or its segments for a model fitted with ‘segment =‘ — from their stored embeddings, with no encoding.
model	A loaded [sbert_model][load_model()], a pinned model name, or ‘NULL‘ for the default model; ignored when ‘embeddings‘ are supplied.
embeddings	Optional numeric matrix with one row per unit.
n	Number of units returned per topic.
rank	‘"margin"‘ (default) or ‘"distance"‘ (raw closeness to the topic centroid).
batch_size	Batch size passed to [encode()] when ‘model‘ is used.

Value

A base data frame with columns ‘topic‘, ‘rank‘, ‘text‘, ‘distance‘ (to the unit's own centroid), and ‘margin‘, ordered by topic and rank. When the fitted units are ranked (‘text = NULL‘), ‘document_id‘ — and ‘segment‘, for a segmented model — sit between ‘rank‘ and ‘text‘ so every example can be traced to its source. Topics to which no unit is assigned contribute no rows.

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)
representatives(
  fitted,
  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 *Segment Text into Sentences, Clauses, or Phrases*

Description

Splits each document into units at a selectable granularity using deterministic sentence-boundary rules: a word-boundary-anchored, case-insensitive abbreviation gazetteer (see [abbreviations()]), a decimal-number guard, and a parenthetical guard, plus shallow clause chunking at discourse connectives. No model call is involved, so segmentation works offline and is fully reproducible.

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	A character vector of documents. Names, when present, are carried into the ‘document_name‘ column.
level	‘"clause"‘ (default), ‘"sentence"‘, or ‘"phrase"‘.
merge_below	Re-join segments shorter than this many words into their neighbor. ‘0‘ (default) disables merging and returns the pure segmentation.
abbreviations	Character vector of abbreviations (each ending in a period) whose periods never end a sentence. Defaults to the built-in gazetteer from [abbreviations()]; matching is case-insensitive.

`cores` Number of forked worker processes used to split documents. Default ‘1‘ (serial). Values above one use ‘parallel::mclapply‘ on Unix-alikes and fall back to serial on Windows or for small inputs; the segmentation is identical regardless of the count.

`max_tokens` Optional cap on segment length, so no segment overruns an encoder’s context window and is silently truncated. ‘NULL‘ (default) leaves segments uncapped. When set, any segment over the budget is re-split at the finest logical boundaries — clause hinges, ‘";"‘, ‘":"‘, ‘" - "‘, and commas (though a comma that would sever a name from its initials, as in ‘"Tomlinson, C. A."‘, does not split) — and the pieces are packed back up to the budget, so a split lands on punctuation wherever possible. A run with no such boundary is chopped further, but even then the break is placed just before a function word (a coordinator, preposition, article, or relative pronoun) near the budget edge rather than mid-phrase, falling back to the raw edge only when the run has no function word either. With ‘model = NULL‘ the budget counts whitespace-delimited words, a deterministic offline proxy; set it below the model’s true token limit (for example around 200 for a 256-token model), since a tokenizer emits somewhat more tokens than words.

`model` Optional loaded [sbert_model][load_model()]. When supplied with ‘max_tokens‘, the budget counts that model’s exact sub-word tokens instead of words, so ‘max_tokens‘ can be the model’s real limit. Token-counted segmentation runs serially (the tokenizer is not forked), so ‘cores‘ is ignored in that case.

`min_content` Minimum alphabetic-content ratio, in ‘[0, 1]‘, for a segment to be kept (see [content_ratio()]). Segments below the floor — citation fragments, page and reference bits, number lists — are dropped. Prose scores near 1 and reference noise far below, so a value around ‘0.5‘ removes the noise while keeping real clauses. ‘0‘ (default) keeps every segment.

Details

Each finer level adds separators:

‘"sentence"‘ splits at ‘.‘, ‘¿, and ‘¡ only.

‘"clause"‘ additionally splits at ‘;‘, ‘:‘, spaced dashes, and subordinating hinges (for example "which", "where", "in terms of"). Comma-separated enumerations stay whole. This is the default.

‘"phrase"‘ additionally splits at commas, for maximal granularity.

Before segmentation, each document is normalized: curly quotes become straight quotes, en and em dashes become spaced hyphens, and runs of whitespace collapse to single spaces. Letter case is preserved.

Value

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

Examples

```
segment(
  "We propose a simulator which runs alongside the processor."
)

segment(
  c(intro = "See Fig. 3 for details. The next part follows."),
  level = "sentence"
)
```

`stop_words` *Obtain and Adjust the Topic Stop-Word List*

Description

Returns the stop words excluded from term extraction and keyword candidates. ‘add‘ extends the list — the standard way to remove a corpus-wide vocabulary from topic labels (words every document shares carry no discriminative information). ‘remove‘ un-lists built-in entries whose surface form is meaningful in a specific corpus.

Usage

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

Arguments

`language`	Currently only ‘"en"‘ is supported.
`add`	Character vector of extra words to exclude (matched case-insensitively).
`remove`	Character vector of words to drop from the list.

Details

For a fully custom list, pass any character vector straight to the ‘stop_words‘ argument of [topics()], [compare_topics()], [topic_corpus()], or [keywords()] — for example ‘stop_words(add = c("covid", "patient"))‘ to keep the defaults plus domain terms, a bare ‘c("covid", "patient")‘ to replace the list entirely, or ‘character()‘ to disable stop-word filtering.

Value

A sorted character vector of lowercase stop words.

Examples

```
head(stop_words())
# keep the defaults and add domain terms
custom <- stop_words(add = c("students", "learning"), remove = "against")

# use a custom list when modeling
text <- c("students learning math", "markets and trading stocks")
embeddings <- rbind(c(1, 0), c(0, 1))
topics(text, n_topics = 2, embeddings = embeddings, stop_words = custom)
```

`strip_list_markers` *Strip Enumeration and List Markers from Text*

Description

Removes list and enumeration markers — ‘"1."‘, ‘"2."‘, ‘"(i)"‘, ‘"(a)"‘, ‘"(iv)"‘, ‘"IV."‘, ‘"V."‘ — from each document. The ‘numbers‘, ‘roman_numerals‘, and ‘section_numbers‘ arguments of [topics()] clean only the extracted *terms*; this cleans the source *text*, so markers no longer over-fragment [segment()], skew the embeddings, or surface in [representatives()][representatives]. Run it once, before embedding or segmentation, as a companion to [dedupe()].

Usage

```
strip_list_markers(text)
```

Arguments

`text`	A character vector.

Details

A marker must stand alone — whitespace or a string boundary on both sides — so real words in parentheses (‘"(civil)"‘), four-digit years (‘"(2020)"‘), counts (‘"(n=100)"‘), multi-word parentheticals (‘"(see Fig 1)"‘), and decimals (‘"1.5"‘) are left untouched. Removed forms are a standalone one- or two-digit number followed by a period (‘"1."‘, ‘"12."‘); a parenthesized one- or two-digit number, single letter, or Roman numeral up to 39 (‘"(1)"‘, ‘"(a)"‘, ‘"(iv)"‘); and a bare upper-case Roman-numeral section marker with a period (‘"II."‘, ‘"IV."‘). A multi-letter numeral is removed anywhere; a single-letter one (‘"V."‘, ‘"X."‘) only at the start of the text, so a mid-sentence initial (‘"by V. Nabokov"‘), unit (‘"5 V."‘), and lower-case ‘"v."‘ (versus) are kept.

Value

‘text‘ with markers removed and runs of whitespace collapsed to a single space; each element trimmed.

See Also

[dedupe()], and the ‘numbers‘ / ‘roman_numerals‘ / ‘section_numbers‘ arguments of [topics()] for cleaning the terms rather than the text.

Examples

```
strip_list_markers(c("1. background 2. methods", "(i) first (ii) second"))
strip_list_markers("the concept of (civil) society in (2020)")
```

---

summary.sbert_topic_model

Summarize a Semantic Topic Model

Description

Prints a compact scientific report – corpus size, cluster separation, coherence, and topic_diversity – and returns a tidy per-topic quality table.

Usage

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

Arguments

`object`	An ‘sbert_topic_model‘ returned by [topics()].
`measure`	Coherence measure passed to [coherence()].
`n_terms`	Number of top terms per topic used for coherence and topic_diversity.
`...`	Unused; present for S3 compatibility.

Value

Invisibly, a data frame with one row per topic containing ‘topic‘, ‘label‘, ‘n_documents‘, ‘proportion‘, and ‘coherence‘.

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))
topics <- topics(text, 2, embeddings = embeddings, n_terms = 3)
summary(topics)
```

terms.sbert_topic_model

Topic Terms, Retuned Without Refitting

Description

Recomputes the ranked terms of a fitted topic model under different term settings. Term extraction depends only on the fitted topic assignments and the document text, so every argument here can be changed without repeating the clustering or the encoding.

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 fitted model from [topics()].

`n` Terms returned per topic, or ‘NULL‘ for the whole vocabulary (useful together with ‘smoothing‘ to obtain a full $p(term|topic)$ distribution). Defaults to the model’s fitted ‘n_terms‘.

`stop_words` Character vector excluded from terms. Defaults to the model’s fitted list; see [stop_words()].

`min_term_frequency` Minimum corpus-wide token frequency.

`min_token_length` Minimum Unicode character length for a token.

`weighting` Class-based term weighting, ‘"ctfidf"‘ or ‘"bm25"‘.

`reduce_frequent_words` Square-root the within-topic frequency before weighting, damping very frequent words.

`stem` Collapse inflected forms onto a shared Porter stem, displaying the most frequent surface form. Requires ‘SnowballC‘.

`numbers` How to treat purely numeric tokens: ‘"keep"‘ or ‘"remove"‘ (drop digit-only tokens such as years and counts). Defaults to the model’s fitted setting.

`roman_numerals` How to treat Roman-numeral tokens: ‘"keep"‘ or ‘"remove"‘ (drop chapter/section markers such as ‘ii‘, ‘iv‘, up to 100). Defaults to the model’s fitted setting.

`section_numbers` How to treat section, reference, and list numbering — multi-level indices such as ‘1.2.3‘ and enumeration markers such as ‘1.‘, ‘2.‘: ‘"keep"‘ or ‘"remove"‘. Defaults to the model’s fitted setting.

`smoothing` Additive (Dirichlet) smoothing for the ‘beta‘ column. Only terms a topic actually used are returned, so with ‘smoothing = 0‘ (default) each topic’s ‘beta‘ sums to one, while any positive value reserves mass for the zero-count vocabulary and the returned rows sum to less than one.

`sort_by` Order terms within each topic by ‘"score"‘ (default, the class-based weight — distinctive words) or ‘"beta"‘ (raw $p(term|topic)$ — the words the topic uses most). ‘n‘ is applied after ordering.

`...` Unused, present for generic consistency.

Value

A base data frame with one row per topic and term: ‘topic‘, ‘label‘, ‘term‘, ‘rank‘, ‘score‘ (the class-based weight), ‘frequency‘ (within-topic count), and ‘beta‘, the multinomial $p(term|topic)$.

Examples

```
text <- c(
  "Cats chase mice", "Kittens chase mice too", "Cats nap daily",
  "Stocks and bonds trade", "Markets price shares", "Banks report profit"
)
embeddings <- rbind(
  c(1, 0, 0), c(0.98, 0.02, 0), c(0.96, 0, 0.04),
  c(0, 1, 0), c(0.02, 0.98, 0), c(0, 0.96, 0.04)
)
fitted <- topics(
  text, n_topics = 2, embeddings = embeddings, min_term_frequency = 1
)
terms(fitted, n = 3)
terms(fitted, n = 3, weighting = "bm25")
```

`topic_corpus` *Prepare a Corpus Once to Fit Many Topic Models*

Description

Runs the corpus-level work that does not depend on the number of topics — segmenting (when ‘segment‘ asks for it), embedding every unit, and tokenizing it for term scoring — a single time, so that fitting many models (a [compare_topics()] sweep, a manual loop over topic counts, or repeated [topics()] calls) reuses it instead of repeating it. Pass the returned object to [topics()] or [compare_topics()] in place of the raw ‘text‘.

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`	Character vector containing one document per element.
`column`	When ‘text‘ is a data frame, the name of the column holding the documents to model. Every other column is carried into ‘$documents‘, and rows whose text is missing, blank, or a bibliographic placeholder are dropped. Leave ‘NULL‘ for a character vector.
`model`	A loaded [sbert_model][load_model()], a pinned model name from [models()], or ‘NULL‘ for the default model. Ignored when ‘embeddings‘ are supplied.
`embeddings`	Optional numeric matrix with one row per document; when supplied, no model is loaded or used.
`batch_size`	Batch size passed to [encode()] when ‘model‘ is used.
`stop_words`	Character vector excluded from topic terms. Use ‘character()‘ to disable stopword filtering.
`min_token_length`	Minimum Unicode character length for a topic token.
`stem`	Whether to collapse inflected forms (for example ‘animals‘ and ‘animal‘) onto a shared Porter stem before scoring, displaying the most frequent surface form of each stem. Requires the ‘SnowballC‘ package.

`cores` Number of forked worker processes used to tokenize documents for term scoring. Default ‘1‘ (serial). Values above one use ‘parallel::mclapply‘ on Unix-alikes and fall back to serial on Windows or for small corpora; the tokenization — and therefore every result — is identical regardless of the count. Ignored when a prepared [topic_corpus()] is supplied (its tokens are already computed).

`numbers` How to treat purely numeric tokens (years, counts) in topic terms. ‘"keep"‘ (default) keeps them; ‘"remove"‘ drops tokens made only of digits, while retaining alphanumerics such as ‘covid19‘.

`roman_numerals` How to treat Roman-numeral tokens (chapter, section, and list markers such as ‘ii‘, ‘iv‘, ‘xii‘). ‘"keep"‘ (default) keeps them; ‘"remove"‘ drops canonical Roman numerals up to 100. The bound is deliberate: larger Roman numerals collide with common abbreviations that are also valid numerals (‘ml‘, ‘mm‘, ‘cc‘, ‘ci‘, ‘cv‘), which are always kept. A few small numerals that are also words (‘iv‘, ‘vi‘, ‘xl‘) are removed when this is on.

`section_numbers` How to treat section, reference, and list numbering. ‘"keep"‘ (default) keeps it; ‘"remove"‘ strips, before tokenizing, both multi-level indices (‘1.2.3‘, ‘4.5.6.7‘ — three or more dot-separated groups) and enumeration or list markers (‘1.‘, ‘2.‘, ‘figure 12.‘ — a standalone one- or two-digit number followed by a period). Decimals (‘3.14‘), four-digit years (‘2020.‘), hyphenated numbers (‘covid-19.‘), and larger counts are left untouched, so genuine values survive even when ‘numbers = "keep"‘. These filter only the topic terms; the document text and its embedding are untouched. To clean the source text itself — list markers, reference noise, junk characters — before encoding, see [clean_corpus()].

`segment` The unit the model is fitted on. ‘"document"‘ (default) embeds each document whole. ‘"sentence"‘, ‘"clause"‘, or ‘"phrase"‘ first splits every document with [segment()] at that level and fits the topics on the segments, so long documents are modeled in full instead of being truncated to the encoder’s context window, and a document can span several topics. The fitted ‘$documents‘ then has one row per segment with its parent ‘document_id‘ and ‘segment‘ position; [topic_sizes()] counts by segment or by document, [topic_gamma()] recovers each document’s topic mixture from the stored segments, and [predict()] segments new text the same way. A precomputed ‘embeddings‘ matrix must then have one row per segment, aligned with ‘segment(text, level = segment, ...)‘.

`max_tokens` Optional cap on segment length, passed to [segment()]. When a ‘model‘ is used, the budget counts that model’s exact tokens; with precomputed ‘embeddings‘ it counts words. Only with a segmented fit.

`merge_below` Re-join segments shorter than this many words into their neighbor, passed to [segment()]. Only with a segmented fit.

`min_content` Minimum alphabetic-content ratio for a segment to be kept, passed to [segment()]. Only with a segmented fit.

Details

The results are byte-identical to calling [topics()] on the raw text: the corpus only *caches* the segmentation, embedding, and tokenization steps, it does not change them. The corpus fixes the embedding source (‘model‘ or ‘embeddings‘), the segmentation (‘segment‘, ‘max_tokens‘,

‘merge_below‘, ‘min_content‘), and the tokenization settings (‘stop_words‘, ‘min_token_length‘, ‘stem‘, ‘numbers‘, ‘roman_numerals‘, ‘section_numbers‘); [topics()] then refuses conflicting overrides for those, while per-model settings (‘n_topics‘, ‘n_terms‘, ‘min_term_frequency‘, ‘weighting‘, ‘reduce_frequent_words‘, ‘seeds‘, ...) are still chosen per call.

Value

An object of class ‘sbert_topic_corpus‘: a list with the prepared ‘text‘ (one element per modeled unit), carried ‘metadata‘, ‘units‘ (the ‘document_id‘, ‘document_name‘, and ‘segment‘ of every unit when the corpus is segmented, otherwise ‘NULL‘), unit ‘embeddings‘, cached ‘token_lists‘, ‘model‘ information, and the fixed segmentation and tokenization ‘settings‘.

See Also

[topics()], [compare_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))
corpus <- topic_corpus(text, embeddings = embeddings)
topics(corpus, n_topics = 2)$topics
compare_topics(corpus, n_topics = 2:3)
```

---

topic_diversity *Measure Topic Diversity*

Description

Reports the proportion of distinct terms among the pooled top terms of every topic. A value near one indicates topics described by largely different vocabulary; a low value indicates redundant topics that repeat the same words (Dieng et al. 2020).

Usage

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

Arguments

object	An ‘sbert_topic_model‘ returned by [topics()].
n_terms	Number of top terms per topic to pool. Capped at the number of terms available for each topic.

Value

A single proportion in ‘(0, 1]‘.

References

Dieng, A. B., Ruiz, F. J. R., and Blei, D. M. (2020). Topic modeling in embedding spaces. TACL.

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))
topics <- topics(text, 2, embeddings = embeddings, n_terms = 3)
topic_diversity(topics)
```

---

topic_gamma *Document-Topic Distributions from Segment Assignments*

Description

Computes ‘gamma‘, the distribution of every document over the fitted topics, by segmenting each document with [segment()], assigning each segment to its nearest topic centroid, and normalizing the per-document segment counts. This yields parameter-free mixed topic_membership: a document that discusses two topics in different sentences receives weight on both, which the single-embedding hard assignment cannot express.

Usage

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

Arguments

`object`	A fitted [topics()] model.
`text`	Either ‘NULL‘, a character vector of documents (segmented here at ‘level‘), or the data frame returned by [segment()] (used as-is). ‘NULL‘ — the default — works for a model fitted with ‘segment = "sentence"‘, ‘"clause"‘, or ‘"phrase"‘: its own segments and their topic assignments are already stored, so the mixture of every fitted document comes back with no encoding at all. Passing your own segmentation keeps every segment option in [segment()] and lets you reuse

one set of segment embeddings across this and other verbs. A data frame must have ‘document_id‘ and ‘text‘ columns; ‘gamma‘ is then computed per ‘document_id‘.

`model` A loaded [sbert_model][load_model()], a pinned model name, or ‘NULL‘ for the default model, used to embed the segments; ignored when ‘embeddings‘ are supplied.

`embeddings` Optional precomputed numeric matrix of segment embeddings, one row per segment. When ‘text‘ is raw documents the rows must align with ‘segment(text, level = level)‘; when ‘text‘ is a [segment()] data frame they must align with its rows — which is automatic, since no re-segmentation happens.

`level` Segmentation granularity passed to [segment()] when ‘text‘ is a character vector: ‘"clause"‘, ‘"sentence"‘, or ‘"phrase"‘. ‘NULL‘ (the default) uses the fitted segmentation — level, ‘max_tokens‘, ‘merge_below‘, and ‘min_content‘ — for a segmented model, so new documents are cut exactly as the fitted ones were, and ‘"clause"‘ for a document-level model. Ignored when ‘text‘ is already a [segment()] data frame.

`batch_size` Batch size passed to [encode()] when ‘model‘ is used.

`cores` Number of forked worker processes used to split documents into segments (passed to [segment()]). Default ‘1‘. Segmenting a large corpus dominates the non-encoding cost, so ‘cores > 1‘ can noticeably speed up sentence- and clause-level gamma; the result is identical for any count. Encoding itself is parallelized separately, via ‘load_model(threads =)‘. Ignored when ‘text‘ is already a [segment()] data frame.

`dedupe_segments` When ‘TRUE‘, encode each distinct segment only once and expand the embeddings back by position, rather than encoding every occurrence. Real corpora are often close to half duplicate segments, so this can roughly halve encoding time (the dominant cost). Only applies when ‘embeddings‘ are not supplied. Assignments match encoding every occurrence up to the ~1e-7 floating-point effect of encoding a smaller batched set, which can flip a rare borderline segment; it is therefore opt-in and off by default. With the static ‘potion-base-8M‘ model, encoding has no batch effect and the result is identical.

`sort_by_length` Passed to [encode()]. Groups segments of similar length into the same batch so the model computes over less padding. Segment lengths vary far more than document lengths — at clause level 60.1 token work is padding in input order against 1.6 the single largest saving available on this verb, measured at 1.9x. Off by default for the same ~1e-7 reason as ‘dedupe_segments‘.

Value

A base data frame with one row per document-topic pair and columns ‘document_id‘, ‘topic‘, ‘gamma‘, and ‘n_segments‘. ‘gamma‘ sums to 1 within each document; documents with no segments contribute no rows.

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)
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 (with any options), 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_hierarchy` *Build the Topic Hierarchy of a Fitted Model*

Description

Agglomeratively clusters the topic centroids by cosine distance and returns the merge tree: which topics are semantic neighbors, in what order they would fuse, and how far apart they are. Use it to judge whether a topic count is too fine (early merges at small heights are near-duplicate topics) and to choose a target count for [reduce_topics()]. Deterministic — no sampling, no seed.

Usage

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

Arguments

`object`	An ‘sbert_topic_model‘ returned by [topics()].
`method`	Agglomeration method passed to [stats::hclust()]. Default ‘"average"‘.

Value

An ‘sbert_topic_hierarchy‘: a list with ‘merges‘ (a tidy data frame with one row per merge — ‘step‘, ‘height‘, and the human-readable ‘left‘ and ‘right‘ branch descriptions) and ‘tree‘ (the underlying ‘hclust‘ object, labeled with topic labels). ‘print()‘ shows the merge table; ‘plot()‘ draws the labeled dendrogram.

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)
)
topics <- topics(text, 3, embeddings = embeddings, n_terms = 3)
topic_hierarchy(topics)
```

---

topic_membership *Soft Topic Membership Probabilities*

Description

Computes fuzzy-c-means-style topic_membership of every document in every topic from cosine distances to the stored centroids, without changing the deterministic hard clustering. In high-dimensional embedding spaces distances concentrate, so the textbook fuzzifier ‘sharpness = 2‘ collapses memberships toward the uniform ‘1/k‘; the default ‘sharpness = 1.15‘ preserves contrast. The topic_membership ranking of topics within a document is invariant to ‘sharpness‘; the probability magnitudes are not, and should be interpreted relative to the chosen value.

Usage

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

Arguments

`object`	A fitted [topics()] model.
`embeddings`	Optional numeric matrix of document embeddings. When omitted, the embeddings stored by ‘topics(keep_embeddings = TRUE)‘ are used.
`sharpness`	Fuzzy-c-means fuzzifier ‘m‘. Must be greater than 1; values close to 1 give crisper memberships.

Value

A base data frame with one row per document-topic pair and columns ‘document_id‘, ‘topic‘, ‘probability‘, and ‘rank‘ (1 is the strongest topic of the document). Probabilities sum to 1 within each document.

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, keep_embeddings = TRUE)
topic_membership(fitted)
```

---

topic_palette *Qualitative Colour Palette for Topics*

Description

A color-blind-friendly qualitative palette used by the package plots.

Usage

```
topic_palette(n)
```

Arguments

n	Number of colors to return.

Value

A character vector of ‘n‘ hex colors.

Examples

```
topic_palette(4)
```

---

topic_similarity *Compute Cosine Similarity*

Description

Compute Cosine Similarity

Usage

```
topic_similarity(x, y = NULL)
```

Arguments

x	Numeric embedding matrix, or one numeric vector.
y	Optional second embedding matrix or vector. When omitted, computes all pairwise similarities among rows of ‘x‘.

Value

A numeric similarity matrix.

Examples

```
x <- rbind(c(1, 0), c(0, 1), c(1, 1))
topic_similarity(x)
```

---

topic_sizes *Topic Sizes on the Distinct and Weighted Scales*

Description

Returns the size of every topic as fitted (distinct units) and, when ‘weights‘ are supplied, on the weighted scale — for example the original row frequencies from [dedupe()]. The gap between ‘proportion‘ and ‘weighted_share‘ measures how template-driven a topic is: a topic whose weighted share far exceeds its distinct share is a small repertoire of heavily repeated texts.

Usage

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

Arguments

object	A fitted [topics()] model.
weights	Optional numeric vector with one non-negative weight per unit on the chosen scale — per fitted document (in document order), typically the ‘n‘ column of [dedupe()], or per segment when ‘by = "segment"‘ on a segmented model.
by	‘"segment"‘ (default; every fitted unit) or ‘"document"‘ (the distinct source documents).

Details

For a model fitted with ‘segment = "sentence"‘, ‘"clause"‘, or ‘"phrase"‘, ‘by‘ picks the scale: ‘"segment"‘ counts the fitted segments, ‘"document"‘ counts the source documents that have at least one segment in the topic. A document that spans several topics is counted in each, so document proportions need not sum to one. On a document-level model the two scales coincide.

Value

A base data frame with one row per topic and columns ‘topic‘, ‘label‘, ‘n_documents‘, and ‘proportion‘, plus ‘n_weighted‘ and ‘weighted_share‘ when ‘weights‘ are supplied.

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)
topic_sizes(fitted, weights = c(10, 1, 1, 1))
```

---

topics *Discover Semantic Topics in Documents*

Description

Performs deterministic document-level topic clustering with Sentence-BERT embeddings. Supply either a loaded ‘model‘ or a precomputed embedding matrix. Topics are summarized with representative documents and class-based TF-IDF terms. This is embedding-based topic discovery, not a probabilistic LDA model.

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
)
```

Arguments

`text` Character vector containing one document per element.

`n_topics` Number of semantic topics. Must be at least two.

`column` When ‘text‘ is a data frame, the name of the column holding the documents to model. Every other column is carried into ‘$documents‘, and rows whose text is missing, blank, or a bibliographic placeholder are dropped. Leave ‘NULL‘ for a character vector.

`model` A loaded [sbert_model][load_model()], a pinned model name from [models()], or ‘NULL‘ for the default model. Ignored when ‘embeddings‘ are supplied.

`embeddings` Optional numeric matrix with one row per document; when supplied, no model is loaded or used.

`batch_size` Batch size passed to [encode()] when ‘model‘ is used.

`iter_max` Maximum deterministic k-means iterations.

`n_terms` Maximum class-based TF-IDF terms returned per topic.

`n_representatives` Maximum representative documents per topic.

`stop_words` Character vector excluded from topic terms. Use ‘character()‘ to disable stop-word filtering.

`min_term_frequency` Minimum corpus-wide token frequency.

`min_token_length` Minimum Unicode character length for a topic token.

`weighting` Class-based term-weighting scheme. ‘"ctfidf"‘ (default) uses ‘tf * log(1 + A / f_x)‘; ‘"bm25"‘ uses the BM25 inverse-frequency variant ‘tf * log(1 + (A - f_x + 0.5) / (f_x + 0.5))‘, which more aggressively down-weights terms shared across topics (Mendonca and Figueira 2025).

`reduce_frequent_words` Whether to square-root the within-topic term frequency before weighting, damping very frequent words.

`stem` Whether to collapse inflected forms (for example ‘animals‘ and ‘animal‘) onto a shared Porter stem before scoring, displaying the most frequent surface form of each stem. Requires the ‘SnowballC‘ package.

`keep_embeddings` Whether to retain normalized document embeddings in the returned object.

`seeds` Optional guided topics: a character vector (or list of character vectors, collapsed with spaces) of seed words or topic descriptions — one element per seeded topic. Seeds are embedded and become the first ‘length(seeds)‘ cluster centroids; remaining topics (up to ‘n_topics‘) are initialized away from the seeds. Seeded topics keep their position — topic i is seed i, never reordered by size — and a named ‘seeds‘ vector names those topics’ labels. With precomputed ‘embeddings‘, supply ‘seed_embeddings‘ as well.

`seed_embeddings`
: Optional matrix of seed embeddings (one row per seed, same dimension as the document embeddings); required when ‘seeds‘ is used together with precomputed ‘embeddings‘.

`fixed_seeds`
: When ‘TRUE‘, seed centroids are frozen and documents are simply assigned to the nearest seed (zero-shot classification into the seeded topics; ‘n_topics‘ must equal the number of seeds, and topics may be empty). When ‘FALSE‘ (default), seeds only initialize the clustering and the data can move the centroids.

`cores`
: Number of forked worker processes used to tokenize documents for term scoring. Default ‘1‘ (serial). Values above one use ‘parallel::mclapply‘ on Unix-alikes and fall back to serial on Windows or for small corpora; the tokenization — and therefore every result — is identical regardless of the count. Ignored when a prepared [topic_corpus()] is supplied (its tokens are already computed).

`numbers`
: How to treat purely numeric tokens (years, counts) in topic terms. ‘"keep"‘ (default) keeps them; ‘"remove"‘ drops tokens made only of digits, while retaining alphanumerics such as ‘covid19‘.

`roman_numerals`
: How to treat Roman-numeral tokens (chapter, section, and list markers such as ‘ii‘, ‘iv‘, ‘xii‘). ‘"keep"‘ (default) keeps them; ‘"remove"‘ drops canonical Roman numerals up to 100. The bound is deliberate: larger Roman numerals collide with common abbreviations that are also valid numerals (‘ml‘, ‘mm‘, ‘cc‘, ‘ci‘, ‘cv‘), which are always kept. A few small numerals that are also words (‘iv‘, ‘vi‘, ‘xl‘) are removed when this is on.

`section_numbers`
: How to treat section, reference, and list numbering. ‘"keep"‘ (default) keeps it; ‘"remove"‘ strips, before tokenizing, both multi-level indices (‘1.2.3‘, ‘4.5.6.7‘ — three or more dot-separated groups) and enumeration or list markers (‘1.‘, ‘2.‘, ‘figure 12.‘ — a standalone one- or two-digit number followed by a period). Decimals (‘3.14‘), four-digit years (‘2020.‘), hyphenated numbers (‘covid-19.‘), and larger counts are left untouched, so genuine values survive even when ‘numbers = "keep"‘. These filter only the topic terms; the document text and its embedding are untouched. To clean the source text itself — list markers, reference noise, junk characters — before encoding, see [clean_corpus()].

`segment`
: The unit the model is fitted on. ‘"document"‘ (default) embeds each document whole. ‘"sentence"‘, ‘"clause"‘, or ‘"phrase"‘ first splits every document with [segment()] at that level and fits the topics on the segments, so long documents are modeled in full instead of being truncated to the encoder’s context window, and a document can span several topics. The fitted ‘$documents‘ then has one row per segment with its parent ‘document_id‘ and ‘segment‘ position; [topic_sizes()] counts by segment or by document, [topic_gamma()] recovers each document’s topic mixture from the stored segments, and [predict()] segments new text the same way. A precomputed ‘embeddings‘ matrix must then have one row per segment, aligned with ‘segment(text, level = segment, ...)‘.

`max_tokens`
: Optional cap on segment length, passed to [segment()]. When a ‘model‘ is used, the budget counts that model’s exact tokens; with precomputed ‘embeddings‘ it counts words. Only with a segmented fit.

`merge_below`
: Re-join segments shorter than this many words into their neighbor, passed to [segment()]. Only with a segmented fit.

min_content Minimum alphabetic-content ratio for a segment to be kept, passed to [segment()]. Only with a segmented fit.

Details

Topic labels are arbitrary cluster identifiers. Interpret them with the ranked ‘terms‘ and ‘representatives‘ tables. Term scores use the class-based TF-IDF weighting ‘tf * log(1 + A / f_x)‘, where ‘tf‘ is the frequency of a term normalized within its topic, ‘f_x‘ is the term’s frequency across all topics, and ‘A‘ is the mean topic length (a real number, matching Mendonca and Figueira (2025), Eq. 1). Set ‘weighting = "bm25"‘ and/or ‘reduce_frequent_words = TRUE‘ for the BM25 and square-root variants (their Eq. 2 to 4). The built-in tokenizer preserves Unicode alphanumeric tokens and internal apostrophes, but does not perform language-specific word segmentation for unspaced CJK text.

Value

An object of class ‘sbert_topic_model‘ containing unit assignments (‘$documents‘, one row per document or per segment), topic summaries, ranked terms, representatives, centers, clustering diagnostics, and the ‘$settings‘ the fit was made with.

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))
topics <- topics(text, 2, embeddings = embeddings)
topics$topics

# Fit on sentences: each document is split first, and the segments are
# what the topics are made of (the embeddings here are one row per
# sentence, so no model download is needed).
documents <- c(
  "Cats chase mice. Stocks and bonds trade.",
  "Dogs chase balls. Markets price shares."
)
sentence_model <- topics(
  documents, 2,
  segment = "sentence",
  embeddings = rbind(c(1, 0), c(0, 1), c(0.9, 0.1), c(0.1, 0.9))
)
sentence_model$documents
topic_sizes(sentence_model, by = "document")
```

Index

∗ **datasets**
    covid, 13
    feedback_translations, 16

abbreviations, 3
as.data.frame.sbert_topic_sweep, 3

blend, 4

cache_dir, 6
cache_size, 6
clean_corpus, 7
coherence, 8
compare_topics, 9
content_ratio, 12
covid, 13

dedupe, 14

encode, 14

feedback_translations, 16
fitted.sbert_topic_sweep, 17

install_runtime, 18

keywords, 18

load_custom, 20
load_model, 21

model_download, 23
model_remove, 24
model_status, 24
models, 25

plot.sbert_topic_hierarchy, 25
plot.sbert_topic_model, 26
plot.sbert_topic_sweep, 28
pool, 29
predict.sbert_topic_model, 30
print.sbert_model, 31
print.sbert_topic_corpus, 32
print.sbert_topic_hierarchy, 32
print.sbert_topic_model, 33
print.sbert_topic_sweep, 34

reduce_topics, 35
representatives, 36

segment, 37
stop_words, 39
strip_list_markers, 40
summary.sbert_topic_model, 41

terms.sbert_topic_model, 42
topic_corpus, 43
topic_diversity, 46
topic_gamma, 47
topic_hierarchy, 49
topic_membership, 50
topic_palette, 51
topic_similarity, 51
topic_sizes, 52
topics, 53