Package 'Rhobots'

August 8, 2026

Title 'BERTopic'-Style Topic Modeling Without 'Python'

Version 0.1.10

Description Implements the 'BERTopic' topic modeling pipeline directly in R: transformer-based sentence embedding, Uniform Manifold Approximation and Projection dimensionality reduction, Hierarchical Density-Based Spatial Clustering of Applications with Noise clustering, and class-based term frequency-inverse document frequency topic extraction - all without any dependency on 'Python', 'conda', or 'reticulate'. Every stage runs in R through 'torch', 'safetensors', 'tok', 'uwot', and 'dbscan'. The package mirrors the accessor API of the original 'Python' package, adds integrated quality metrics and hyperparameter search tools, and introduces part-of-speech filtered and C-value-ranked representation models.

License MIT + file LICENSE

URL https://github.com/JPvdP/RhoBots

BugReports https://github.com/JPvdP/RhoBots/issues

SystemRequirements Microsoft Visual C++ Redistributable 2022 (Windows only; required by torch)

Depends R (>= 4.1.0)

Encoding UTF-8

RoxygenNote 7.3.3

Imports torch, safetensors, hfhub, tok, jsonlite, uwot, dbscan, Matrix, stats, wordpiece, Rcpp

LinkingTo Rcpp

Suggests testthat (>= 3.0.0), plotly, httr2, gutenbergr, udpipe

Config/testthat/edition 3

NeedsCompilation yes

Author J.P.G. van der Pol [aut, cre]

Maintainer J.P.G. van der Pol <j.p.g.vanderpol@uu.nl>

Repository CRAN

Date/Publication 2026-08-08 14:30:02 UTC

Contents

`agglomerative_clustering` *Agglomerative (hierarchical) clustering*

Description

Wraps `stats::hclust` + `stats::cutree`. Like k-means, every document is assigned to a cluster (no noise label).

Usage

```
agglomerative_clustering(k, linkage = "ward.D2")
```

Arguments

`k`	Number of clusters to cut the dendrogram into.
`linkage`	Linkage method passed to `stats::hclust` (default `"ward.D2"`).

Value

An `agglomerative_clustering` model object.

Examples

```
m <- agglomerative_clustering(k = 5L)
```

`apply_mmr` *Refine topic representations with Maximal Marginal Relevance (MMR)*

Description

After fitting a topic model, the top terms for each topic are selected purely by c-TF-IDF score, which can produce redundant terms (e.g. *model*, *models*, *modeling*). MMR re-ranks the candidate terms by jointly maximising their relevance to the topic and their diversity from already-selected terms.

Usage

```
apply_mmr(
  fit,
  encoder,
  diversity = 0.1,
  top_n = NULL,
  top_n_candidates = NULL,
  verbose = TRUE
)
```

Arguments

`fit`	A `bertopic_fit` object from `fit_bertopic`.
`encoder`	An encoder from `load_hf_bert`, used to embed candidate terms.
`diversity`	Controls the relevance-diversity trade-off. `0.0` gives pure relevance (close to the original c-TF-IDF ranking); `1.0` gives maximum diversity. Default `0.1`.
`top_n`	Number of terms to keep per topic after MMR. Defaults to `fit$top_n_terms`.
`top_n_candidates`	Size of the candidate pool drawn from c-TF-IDF before MMR selection. Must be $\geq$ `top_n`. Increasing this gives MMR a broader pool to diversify from. Defaults to `top_n` (matching Python BERTopic's default behaviour).
`verbose`	Print progress messages (default `TRUE`).

Details

The algorithm mirrors Python BERTopic's `MaximalMarginalRelevance` representation model:

1. For each topic, take the top `top_n_candidates` terms from c-TF-IDF as the candidate pool.
2. Embed every unique candidate term and a *topic reference string* (the candidates joined into one string) using `encoder`.
3. Greedily select `top_n` terms by

$$MMR_i = (1 - \lambda)\,\text{sim}(w_i, \text{topic}) - \lambda \max_{s \in S} \text{sim}(w_i, s)$$

   where S is the set of already-selected terms and λ = `diversity`.
4. Original c-TF-IDF scores are preserved for the selected terms; only the selection and ranking change.

Note: MMR operates on the existing `topic_terms` in the fit object. If you subsequently call `reduce_topics`, `merge_topics`, or `reduce_outliers`, topic terms are recomputed from c-TF-IDF and MMR needs to be re-applied.

Value

An updated `bertopic_fit` with `topic_terms` and `topic_labels` replaced by the MMR-selected representation.

Examples

```
## Not run:
  enc <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  fit <- fit_bertopic(docs = abstracts, encoder = enc)
  fit <- apply_mmr(fit, encoder = enc, diversity = 0.3)
  get_topic(fit, 0L)

## End(Not run)
```

---

build_dtm *Build a sparse document-term matrix from a character vector*

Description

Tokenizes each document (lowercase, split on non-alphanumeric), removes stopwords, optionally generates n-grams, then applies document-frequency filtering. Returns a sparse matrix suitable for the c-TF-IDF step.

Usage

```
build_dtm(
  docs,
  min_df = 2,
  max_df_frac = 0.95,
  stopwords = character(),
  ngram_range = c(1L, 1L)
)
```

Arguments

docs	Character vector of documents.
min_df	Minimum document frequency: terms appearing in fewer than this many documents are dropped.
max_df_frac	Maximum document-frequency fraction: terms appearing in more than this fraction of documents are dropped.
stopwords	Character vector of tokens to remove before n-gram construction.
ngram_range	Integer vector of length 2, e.g. `c(1, 2)` to include both unigrams and bigrams. Default `c(1, 1)` (unigrams only).

Value

A sparse `dgCMatrix` of dimensions `length(docs)` x vocab size, with `colnames` set to the vocabulary.

Examples

```
docs <- c("the cat sat on the mat", "the dog chased the cat",
          "a cat and a dog", "the mat is on the floor")
build_dtm(docs, min_df = 1L)
```

`classify_texts` *Classify or label texts with a fine-tuned BERT-family model*

Description

S3 generic that dispatches on the classifier class returned by `load_hf_classifier()`.

Usage

```
classify_texts(classifier, texts, ...)

## S3 method for class 'hf_classifier'
classify_texts(
  classifier,
  texts,
```

```
  batch_size = 32L,
  max_length = 512L,
  verbose = FALSE,
  ...
)

## Default S3 method:
classify_texts(classifier, texts, ...)
```

Arguments

classifier	An hf_classifier from load_hf_classifier().
texts	A character vector of strings.
...	Unused (for future extension).
batch_size	Number of texts per forward pass. Default 32.
max_length	Maximum token sequence length (including special tokens). Sequences are truncated to this value. Default 512.
verbose	Print batch progress. Default FALSE.

Details

Sequence classification (sentiment, topic, ...): returns a data.frame with columns text, label, score, plus one probability column per label. For problem_type = "regression" the label columns contain raw numeric scores. For problem_type = "multi_label_classification" each label column contains a sigmoid probability and there is no single label or score column.

Token classification / NER: returns a named list of data.frames, one per input text, each with columns token, label, score. Special tokens ([CLS], [SEP], padding) are automatically excluded.

Value

For sequence tasks: a data.frame with nrow(texts) rows. For NER: a list of data.frames, one per input text.

Examples

```
## Not run:
  clf  <- load_hf_classifier("cardiffnlp/twitter-xlm-roberta-base-sentiment")
  res  <- classify_texts(clf, c("I love this!", "Terrible experience."))
  res$label   # c("positive", "negative")
  res$score   # highest class probability

  # VAD regression
  clf <- load_hf_classifier("RobroKools/vad-bert")
  classify_texts(clf, c("I am ecstatic!"))
  # returns: text | valence | arousal | dominance

  # NER
  clf <- load_hf_classifier("dslim/bert-base-NER")
  result <- classify_texts(clf, c("Marie Curie was born in Warsaw."))
```

```
  result[[1]]   # token | label | score

## End(Not run)
```

`cls_pool` *CLS-token pooling: extract the CLS hidden state as the sentence vector*

Description

Returns the first token's hidden state `(B, H)` from a `(B, L, H)` tensor. Used for models whose `1_Pooling/config.json` sets `pooling_mode_cls_token = true` (e.g. some BGE and GTE variants).

Usage

```
cls_pool(hidden)
```

Arguments

`hidden`	A 3-D `torch_tensor` of shape `(batch, seq_len, hidden)`.

Value

A 2-D `torch_tensor` of shape `(batch, hidden)`.

Examples

```
## Not run:
  enc    <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  tokens <- enc$tokenizer$encode_batch(c("hello world"))
  ids    <- torch::torch_tensor(matrix(tokens[[1L]]$ids, nrow = 1L),
                                dtype = torch::torch_long())
  mask   <- torch::torch_tensor(matrix(tokens[[1L]]$attention_mask, nrow = 1L),
                                dtype = torch::torch_long())
  hidden <- enc$model(ids, mask)
  cls_pool(hidden)

## End(Not run)
```

`cluster_docs` *Fit a clustering model and return cluster labels*

Description

Fit a clustering model and return cluster labels

Usage

```
cluster_docs(model, X, seed = 42L)
```

Arguments

`model`	A clustering model object.
`X`	Numeric matrix to cluster.
`seed`	Random seed (default 42).

Value

A list with `$labels` (integer vector; `-1` = noise, `0`, `1`, `2`, `...` = cluster IDs) and `$model` (the fitted model).

Examples

```
## Not run:
  m   <- hdbscan_clustering(min_pts = 3L)
  enc <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  emb <- embed_texts(enc, c("cats and dogs", "machine learning",
                            "pets and animals", "neural networks"))
  res <- cluster_docs(m, emb)
  res$labels

## End(Not run)
```

`compare_topics` *Compare topic prevalence across groups*

Description

Given a grouping variable (e.g. publication year, country, experimental condition) of the same length as the corpus, computes which topics are over- or under-represented in each group relative to a null model of independence.

Usage

```
compare_topics(
  fit,
  groups,
  method = c("chi2", "log_ratio"),
  min_count = 5L,
  verbose = TRUE
)
```

Arguments

`fit`	A `bertopic_fit` object from `fit_bertopic`.
`groups`	Character or factor vector, one entry per document, defining the group membership. Noise documents (`-1`) are excluded from the analysis but the vector still needs one entry per document.
`method`	Test statistic: `"chi2"` (default) or `"log_ratio"`.
`min_count`	Minimum expected count for a (topic, group) cell to be included. Cells below this threshold are silently dropped (default 5).
`verbose`	Print the contingency table (default `TRUE`).

Details

Two statistics are available:

`"chi2"` Signed chi-square contribution: $sign(O-E)\cdot\sqrt{(O-E)^2/E}$. Positive = over-represented, negative = under-represented. A global chi-square test is also reported.

`"log_ratio"` Laplace-smoothed $\log_2$ ratio of observed to expected proportion. Values above 1 mean the topic is twice as prevalent in that group as expected; below -1 means half as prevalent.

Value

A list of class `compare_topics_result` with elements:

`result` Tidy data frame with columns `Topic`, `Name`, `Group`, `Observed`, `Expected`, `Stat`.

`table` Raw contingency table (topics x groups).

`global_statistic`, `global_pvalue` Global chi-square statistic and p-value (`NA` when `method = "log_ratio"`).

See Also

`visualize_comparison`

Examples

```
## Not run:
  enc    <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  fit    <- fit_bertopic(docs = abstracts, encoder = enc)
  groups <- sample(c("A", "B"), length(abstracts), replace = TRUE)
  comp   <- compare_topics(fit, groups)
  visualize_comparison(comp)

## End(Not run)
```

---

cvalue_representation *Construct a C-value representation model*

Description

Use as the `representation_model` argument in `fit_bertopic` to select the n-gram vocabulary using the C-value algorithm (Frantzi et al. 2000) rather than plain document-frequency filtering. Unigrams are always kept alongside the C-value-selected multi-word terms.

Usage

```
cvalue_representation(max_n = 4L, threshold = 0, min_freq = 2L)
```

Arguments

`max_n`	Maximum n-gram length to evaluate (default 4). Longer n-grams are more informative but also rarer; values of 3 - 5 cover most terminology.
`threshold`	Minimum C-value score for a multi-word term to be retained (default 0). Increase to surface only well-established compound terms.
`min_freq`	Minimum corpus frequency for a candidate n-gram to be considered (default 2).

Value

An object of class `cvalue_representation`.

See Also

`cvalue_terms`, `fit_bertopic`, `pos_representation`

Examples

```
m <- cvalue_representation(max_n = 3L, threshold = 0.5)
```

`cvalue_terms` *Compute C-value scores for candidate multi-word terms*

Description

Implements the C-value method of Frantzi, Ananiadou & Mima (2000). For each candidate n-gram t (n >= 2):

Usage

```
cvalue_terms(
  docs,
  max_n = 4L,
  min_freq = 2L,
  threshold = 0,
  stopwords = character(0L)
)
```

Arguments

`docs`	Character vector of documents.
`max_n`	Maximum n-gram length to consider (default 4).
`min_freq`	Minimum corpus frequency for a candidate term (default 2).
`threshold`	Minimum C-value to include in the returned table (default 0).
`stopwords`	Character vector of tokens to remove before n-gram extraction (default none).

Details

$$C\text{-value}(t) = \log_2 |t| \times \begin{cases} f(t) & \text{if } P(t) = \emptyset \\ f(t) - \dfrac{1}{|P(t)|} \displaystyle\sum_{a \in P(t)} f(a) & \text{otherwise} \end{cases}$$

where $|t|$ is word count, $f(t)$ is corpus frequency, and $P(t)$ is the set of longer candidate terms that contain t as a contiguous sub-sequence.

Value

A data frame with columns `term` (underscore-separated tokens), `freq`, `n_words`, and `cvalue`, sorted by descending C-value. Terms with `cvalue` < `threshold` are excluded.

References

Frantzi, K., Ananiadou, S., & Mima, H. (2000). Automatic recognition of multi-word terms: the C-value/NC-value method. *International Journal on Digital Libraries*, 3(2), 115 - 130.

See Also

cvalue_representation, fit_bertopic

Examples

```
docs <- c("sea level rise and climate change", "sea level is rising",
          "climate change impacts sea level", "Arctic ice melt sea level")
cvalue_terms(docs, max_n = 3L, min_freq = 2L)
```

---

c_tf_idf *Class-based TF-IDF (c-TF-IDF) for cluster-level topic terms*

Description

Treats each cluster as one big document and ranks terms by class TF multiplied by a class-based inverse document frequency. This is the BERTopic-flavoured TF-IDF that produces interpretable topic descriptors.

Usage

```
c_tf_idf(dtm, cluster_ids, top_n = 10, reduce_frequent_words = FALSE)
```

Arguments

dtm	A sparse document-term matrix from build_dtm().
cluster_ids	Integer vector of cluster assignments, one per row of dtm. Use -1 for noise/unassigned documents.
top_n	Number of top terms to return per cluster.
reduce_frequent_words	If TRUE, apply a square-root to the class TF before multiplying by IDF. This down-weights terms that are very frequent within a class and often improves topic interpretability. Mirrors the reduce_frequent_words option in Python BERTopic's ClassTfidfTransformer. Default FALSE.

Value

A data frame with columns topic, rank, term, score.

Examples

```
docs <- c("the cat sat on mat", "a dog chased cat",
          "machine learning models", "deep learning neural nets")
dtm  <- build_dtm(docs, min_df = 1L)
c_tf_idf(dtm, cluster_ids = c(0L, 0L, 1L, 1L), top_n = 3L)
```

dim_project *Project new data using a fitted dimensionality-reduction model*

Description

Project new data using a fitted dimensionality-reduction model

Usage

```
dim_project(model, X)
```

Arguments

model	A fitted dimensionality-reduction model (returned inside the list from dim_reduce).
X	New numeric matrix to project.

Value

A numeric matrix with the same number of columns as the training embedding.

Examples

```
## Not run:
  m    <- umap_reduction(n_neighbors = 5L, n_components = 2L)
  enc  <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  emb  <- embed_texts(enc, c("first doc", "second doc", "third doc"))
  res  <- dim_reduce(m, emb)
  new_emb <- embed_texts(enc, c("new document"))
  dim_project(res$model, new_emb)

## End(Not run)
```

dim_reduce *Fit a dimensionality-reduction model and return the reduced matrix*

Description

Fit a dimensionality-reduction model and return the reduced matrix

Usage

```
dim_reduce(model, X, seed = 42L, verbose = FALSE)
```

Arguments

model	A dimensionality-reduction model object.
X	Numeric matrix to reduce.
seed	Random seed (default 42).
verbose	Print progress (default FALSE).

Value

A list with $embedding (reduced matrix) and $model (the fitted model, for use with dim_project).

Examples

```
## Not run:
  m   <- umap_reduction(n_neighbors = 5L, n_components = 2L)
  enc <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  emb <- embed_texts(enc, c("first doc", "second doc", "third doc"))
  res <- dim_reduce(m, emb)

## End(Not run)
```

---

embed_texts.api_embedder

Embed a vector of texts to a numeric matrix

Description

S3 generic dispatching on the encoder class. For local BERT-family models (class bert_encoder) see the Details section. For API-backed models (class api_embedder) the function calls the provider's REST endpoint and parameters max_length, device, chunk_strategy, and chunk_overlap are ignored.

Usage

```
## S3 method for class 'api_embedder'
embed_texts(
  encoder,
  texts,
  batch_size = NULL,
  normalize = TRUE,
  prefix = NULL,
  verbose = interactive(),
  ...
)

embed_texts(encoder, texts, ...)
```

```
## S3 method for class 'bert_encoder'
embed_texts(
  encoder,
  texts,
  batch_size = 32L,
  max_length = 256L,
  normalize = TRUE,
  device = "cpu",
  prefix = NULL,
  chunk_strategy = c("truncate", "mean", "first"),
  chunk_overlap = 0L,
  verbose = interactive(),
  ...
)

## Default S3 method:
embed_texts(encoder, texts, ...)
```

Arguments

encoder	A loaded encoder: a bert_encoder from load_hf_bert() / load_specter2(), or an api_embedder from load_openai_embedder() / load_cohere_embedder().
texts	A character vector of strings to embed.
batch_size	Number of texts (or chunks) per forward pass.
normalize	If TRUE (default), L2-normalize each row so cosine similarity equals dot product.
prefix	String prepended to every text before tokenization. NULL (default) falls back to encoder$prefix; "" disables prefixing even when the encoder has a stored prefix.
verbose	If TRUE, prints batch progress.
...	Not used; retained for S3 method compatibility.
max_length	Truncate/chunk token sequences to this length (including special tokens). Capped at the model's max_position_embeddings.
device	Either "cpu" (default) or "cuda".
chunk_strategy	One of "truncate" (default), "mean", or "first". See Details. Ignored for api_embedder.
chunk_overlap	Number of token overlap between consecutive windows when chunk_strategy != "truncate". Default 0.

Details

Tokenizes, runs the encoder forward pass, pools over tokens (mean or CLS, auto-detected from the encoder's pooling configuration), and optionally L2-normalizes the result. Batches inputs for memory efficiency.

An instruction **prefix** can be prepended to every text before tokenization. This is required for best performance with BGE and E5 models:

- BGE (BAAI/bge-*): prefix = "Represent this sentence: "
- E5 (intfloat/e5-*): prefix = "passage: "

If the encoder was loaded with a prefix via load_hf_bert()'s prefix argument, that value is used automatically and need not be repeated here. Passing prefix explicitly always takes precedence.

Long-document chunking (chunk_strategy): BERT-family models have a fixed maximum sequence length (usually 512 tokens) and silently truncate longer inputs. Setting chunk_strategy = "mean" (or "first") instead splits each over-length text into overlapping windows of max_length tokens, embeds each window, and aggregates:

- "truncate" (default): truncate at max_length, fast, no overhead.
- "mean": embed all windows, average their vectors (then normalize if normalize = TRUE). Best quality for long documents.
- "first": embed only the first window. Good when the lead of a document (abstract, summary) is the most informative part.

Use chunk_overlap to set the number of overlapping tokens between consecutive windows (default 0).

Value

A numeric matrix with length(texts) rows and hidden_size cols.

Examples

```
## Not run:
  # General model  --  no prefix needed
  enc <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  emb <- embed_texts(enc, c("First sentence.", "Second sentence."))

  # BGE model  --  set prefix at load time (applied automatically)
  enc <- load_hf_bert("BAAI/bge-base-en-v1.5",
                      prefix = "Represent this sentence: ")
  emb <- embed_texts(enc, docs)

  # E5 model  --  passage prefix for documents, query prefix for queries
  enc <- load_hf_bert("intfloat/e5-base-v2", prefix = "passage: ")
  doc_emb   <- embed_texts(enc, docs)
  query_emb <- embed_texts(enc, queries, prefix = "query: ")

  # Long documents: chunk and mean-pool windows
  enc <- load_hf_bert("sentence-transformers/all-mpnet-base-v2")
  emb <- embed_texts(enc, long_papers, chunk_strategy = "mean",
                     chunk_overlap = 32L)

  # API-based (no torch required)
  enc <- load_openai_embedder()
  emb <- embed_texts(enc, docs)

## End(Not run)
```

`embed_texts_cached` *Compute or load document embeddings from a cache file*

Description

On the first call (or when `overwrite = TRUE`) the encoder is used to compute embeddings and the result is written to `cache_file`. On subsequent calls the cached matrix is read directly, making it free to experiment with different dimensionality-reduction or clustering settings without re-running the expensive forward passes.

Usage

```
embed_texts_cached(
  encoder = NULL,
  texts,
  cache_file = NULL,
  overwrite = FALSE,
  batch_size = 32L,
  max_length = 256L,
  normalize = TRUE,
  device = "cpu",
  prefix = NULL,
  chunk_strategy = c("truncate", "mean", "first"),
  chunk_overlap = 0L,
  verbose = interactive()
)
```

Arguments

`encoder`	An encoder from `load_hf_bert`. Required when computing embeddings; may be `NULL` when loading from cache.
`texts`	Character vector of documents to embed. Still required even when loading from cache so the row count can be validated.
`cache_file`	Path to a `.rds` file. If the file exists and `overwrite = FALSE`, embeddings are loaded from it. If the file does not exist (or `overwrite = TRUE`), embeddings are computed and written to this path. Pass `NULL` to skip caching entirely.
`overwrite`	If `TRUE`, always recompute even if the cache file exists (default `FALSE`).
`batch_size, max_length, normalize, device, prefix, chunk_strategy, chunk_overlap`	Forwarded to `embed_texts`. `prefix` defaults to `NULL`, inheriting the encoder's stored prefix.
`verbose`	Print progress messages.

Value

A numeric matrix with `length(texts)` rows.

Examples

```
## Not run:
  enc <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  docs <- c("First document.", "Second document.", "Third document.")
  emb  <- embed_texts_cached(enc, docs, cache_file = tempfile(fileext = ".rds"))

## End(Not run)
```

`find_topics` *Find topics most similar to a search term*

Description

Ranks non-noise topics by the c-TF-IDF score of `search_term`. If the term is absent from the vocabulary entirely, falls back to substring matching across each topic's top terms.

Usage

```
find_topics(fit, search_term, top_n = 5L)
```

Arguments

`fit`	A `bertopic_fit` object from `fit_bertopic`.
`search_term`	A single character string.
`top_n`	Number of topics to return (default 5).

Value

A data frame with columns `Topic`, `Name`, `Score`, sorted by descending score.

Examples

```
## Not run:
  enc <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  fit <- fit_bertopic(docs = abstracts, encoder = enc)
  find_topics(fit, "neural network")

## End(Not run)
```

`fit_bertopic` *Fit a BERTopic-style topic model*

Description

Runs the four-stage BERTopic pipeline:

1. Embed documents with the supplied encoder.
2. Reduce embedding dimensionality with UMAP.
3. Cluster the reduced space with HDBSCAN.
4. Extract per-topic terms with c-TF-IDF.

Usage

```
fit_bertopic(
  encoder = NULL,
  docs,
  embeddings = NULL,
  dim_reduction_model = NULL,
  cluster_model = NULL,
  representation_model = NULL,
  umap_n_neighbors = 15,
  umap_n_components = 5,
  umap_min_dist = 0,
  umap_metric = "cosine",
  hdbscan_min_pts = 10,
  hdbscan_method = c("eom", "leaf"),
  top_n_terms = 10,
  language = "english",
  stopwords = NULL,
  extra_stopwords = NULL,
  ngram_range = c(1L, 1L),
  reduce_frequent_words = FALSE,
  seed = 42,
  verbose = TRUE
)
```

Arguments

`encoder`	A loaded encoder, as returned by `load_hf_bert()`.
`docs`	Character vector of documents.
`embeddings`	Optional pre-computed embedding matrix (rows = documents, columns = dimensions). When supplied, `encoder` is ignored for embedding and its value may be `NULL`.

`dim_reduction_model`
: A dimensionality-reduction model object from `umap_reduction()`, `pca_reduction()`, or `no_reduction()`. When `NULL` (default), a UMAP model is built from the legacy `umap_*` parameters.

`cluster_model`
: A clustering model from `hdbscan_clustering()`, `kmeans_clustering()`, or `agglomerative_clustering()`. When `NULL` (default), HDBSCAN is used with the legacy `hdbscan_*` parameters.

`representation_model`
: Optional representation model from `cvalue_representation()`, `pos_representation()`, or similar. When `NULL` (default) standard c-TF-IDF is used.

`umap_n_neighbors`
: UMAP `n_neighbors` parameter (default 15).

`umap_n_components`
: Reduced dimensionality for clustering (default 5).

`umap_min_dist`
: UMAP `min_dist` (default 0).

`umap_metric`
: UMAP distance metric (default `"cosine"`).

`hdbscan_min_pts`
: HDBSCAN `minPts` (default 10).

`hdbscan_method`
: HDBSCAN cluster-extraction method: `"eom"` (excess of mass, default) or `"leaf"`.

`top_n_terms`
: Number of terms per topic in the output (default 10).

`language`
: Language for built-in stopword list (default `"english"`). Passed to `get_stopwords()`.

`stopwords`
: Character vector of words to drop before c-TF-IDF. Replaces the built-in list when supplied.

`extra_stopwords`
: Additional stopwords appended to the built-in (or user-supplied) list.

`ngram_range`
: Integer vector of length 2 specifying the minimum and maximum n-gram sizes for c-TF-IDF (default `c(1L, 1L)` = unigrams only).

`reduce_frequent_words`
: If `TRUE`, apply a square-root IDF dampening to very frequent words (default `FALSE`).

`seed`
: Random seed for reproducibility (default 42).

`verbose`
: Whether to print progress messages (default TRUE).

Details

Also computes a separate 2-D UMAP projection of the same embeddings, for visualization.

This implementation will be extended over time to mirror more of the Python BERTopic package's capabilities (custom vectorizers, topic reduction, representation models, dynamic topic modeling).

Value

A list (class `bertopic_fit`) with elements: `embeddings`, `reduced`, `layout2d`, `clusters`, `topic_terms`, `hdbscan`, `docs`.

Examples

```
## Not run:
  enc <- load_hf_bert("pritamdeka/S-Scibert-snli-multinli-stsb")
  fit <- fit_bertopic(enc, docs = my_abstracts)
  print_topics(fit)

## End(Not run)
```

`fit_topics_over_time` *Fit independent BERTopic models per time period and align topics*

Description

All documents are embedded once using the shared encoder, then one BERTopic model is fitted per time period using the appropriate slice of embeddings. Because every model lives in the same embedding space, cosine similarity between topic centroids across periods is a meaningful measure of topic continuity.

Usage

```
fit_topics_over_time(
  encoder,
  docs,
  timestamps,
  min_similarity = 0.7,
  bertopic_params = list(),
  verbose = TRUE
)
```

Arguments

`encoder`	An encoder from `load_hf_bert`. Used to embed all documents once before running per-period models.
`docs`	Character vector of all documents.
`timestamps`	A vector of the same length as `docs` assigning each document to a time period. Unique values (sorted) define the period order.
`min_similarity`	Minimum cosine similarity between topic centroids in adjacent periods to count as a link (default `0.7`).
`bertopic_params`	A named list of additional arguments passed to `fit_bertopic` for every period (e.g. `list(hdbscan_min_pts = 5, ngram_range = c(1L, 2L))`).
`verbose`	Print progress messages (default `TRUE`).

Details

Adjacent periods are compared by computing the full cosine-similarity matrix between their topic centroids. Any pair whose similarity exceeds `min_similarity` becomes a directed link in the transition graph. From that graph each topic is labelled:

`emerges` No incoming link from the previous period.

`disappears` No outgoing link to the next period.

`continues` Exactly one incoming and one outgoing link.

`splits` One incoming link, multiple outgoing links.

`merges` Multiple incoming links, one outgoing link.

`isolated` No links in either direction.

Value

An object of class `bertopic_flow` containing:

`periods` Character vector of period labels in order.

`fits` Named list of `bertopic_fit` objects.

`transitions` Data frame of inter-period topic links: `period_from`, `topic_from`, `period_to`, `topic_to`, `similarity`.

`topic_info` Data frame with one row per (period, topic): `period`, `topic`, `label`, `count`, `status`.

Examples

```
## Not run:
  enc    <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  period <- rep(c("2010s", "2020s"), each = 50L)
  flow   <- fit_topics_over_time(enc, docs = abstracts,
                                 period_var = period)
  visualize_topic_flow(flow)

## End(Not run)
```

---

get_document_info *Get document-level topic assignments as a data frame*

Description

Get document-level topic assignments as a data frame

Usage

```
get_document_info(fit, metadata = NULL)
```

Arguments

fit	A bertopic_fit object from fit_bertopic.
metadata	An optional named list of additional columns to append, each with length(fit$docs) elements.

Value

A data frame with one row per document and columns Document, Topic, Name, Top_words.

Examples

```
## Not run:
  enc <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  fit <- fit_bertopic(docs = abstracts, encoder = enc)
  get_document_info(fit)

## End(Not run)
```

---

get_representative_docs

Get representative documents for one or all topics

Description

Representative documents are the three training documents whose embeddings lie closest (cosine similarity) to the topic centroid.

Usage

```
get_representative_docs(fit, topic = NULL)
```

Arguments

fit	A bertopic_fit object from fit_bertopic.
topic	Optional integer topic ID. When NULL (default), returns a named list for all non-noise topics.

Value

A character vector (single topic) or a named list of character vectors (all topics).

Examples

```
## Not run:
  enc <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  fit <- fit_bertopic(docs = abstracts, encoder = enc)
  get_representative_docs(fit, topic = 0L)

## End(Not run)
```

get_stopwords *Return the built-in stopword list for a language*

Description

Currently only "english" is supported; returns an empty character vector for any other value.

Usage

```
get_stopwords(language = "english")
```

Arguments

language	Language name (default "english").

Value

A character vector of stopwords.

Examples

```
get_stopwords("english")
```

get_topic *Get term-score representation for a single topic*

Description

Get term-score representation for a single topic

Usage

```
get_topic(fit, topic)
```

Arguments

fit	A bertopic_fit object from fit_bertopic.
topic	Integer topic ID.

Value

A data frame with columns term and score, or NULL if the topic ID is not found.

Examples

```
## Not run:
  enc <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  fit <- fit_bertopic(docs = abstracts, encoder = enc)
  get_topic(fit, topic = 0L)

## End(Not run)
```

`get_topics` *Get all topic-term representations*

Description

Get all topic-term representations

Usage

```
get_topics(fit, top_n = NULL)
```

Arguments

`fit`	A `bertopic_fit` object from `fit_bertopic`.
`top_n`	Maximum terms per topic. `NULL` (default) returns all stored terms.

Value

A named list; each element is a data frame with columns `term` and `score` sorted by descending score. Names are topic IDs (character strings, including `"-1"` for noise).

Examples

```
## Not run:
  enc <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  fit <- fit_bertopic(docs = abstracts, encoder = enc)
  get_topics(fit, top_n = 5L)

## End(Not run)
```

get_topic_info *Get topic-level metadata as a data frame*

Description

Get topic-level metadata as a data frame

Usage

```
get_topic_info(fit, topic = NULL)
```

Arguments

fit	A bertopic_fit object from fit_bertopic.
topic	Optional integer. If provided, only the row for that topic is returned.

Value

A data frame with columns:

Topic Integer topic ID (-1 = noise/unassigned).

Count Number of documents assigned to this topic.

Name Auto-generated label (e.g. "0_model_data_...").

Representation Top-5 terms, comma-separated.

Examples

```
## Not run:
  enc <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  fit <- fit_bertopic(docs = abstracts, encoder = enc)
  get_topic_info(fit)

## End(Not run)
```

guided_fit_bertopic *Guided topic modeling with user-supplied seed words*

Description

Extends fit_bertopic with a seed_topic_list argument. For each seed topic the user supplies a character vector of representative words; these are concatenated into virtual "anchor" documents, embedded, and added to the corpus before dimensionality reduction and clustering. The anchors bias the latent space so that documents semantically close to the seed words cluster together. Seed documents are removed from the fit before the object is returned.

Usage

```
guided_fit_bertopic(
  docs,
  seed_topic_list,
  embeddings = NULL,
  encoder = NULL,
  n_anchor_weight = 3L,
  ...,
  verbose = TRUE
)
```

Arguments

docs	Character vector of documents.
seed_topic_list	Named list of character vectors – one entry per expected topic. Example: `list("Energy" = c("solar", "wind", "renewable", "battery"), "Economics" = c("labour", "wages", "capital", "trade"))`.
embeddings	Optional pre-computed embedding matrix for `docs`. The encoder is still required to embed the seed words.
encoder	An encoder from `load_hf_bert`, used to embed both documents (when `embeddings` is `NULL`) and seed words.
n_anchor_weight	Number of times each anchor document is replicated before adding to the corpus. Higher values give seeds more influence on the UMAP layout and HDBSCAN clustering (default 3).
...	Additional arguments forwarded to `fit_bertopic` (e.g. `umap_n_neighbors`, `hdbscan_min_pts`).
verbose	Print progress messages (default `TRUE`).

Value

A `bertopic_fit` object. The attribute `seed_map` records which cluster each seed topic was assigned to.

See Also

`fit_bertopic`, `zero_shot_topics`

Examples

```
## Not run:
  enc   <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  seeds <- list(
    "Climate" = c("carbon", "emissions", "greenhouse", "warming"),
    "Economy" = c("trade", "wages", "labour", "inflation")
  )
  fit <- guided_fit_bertopic(abstracts, seed_topic_list = seeds, encoder = enc)
```

```
## End(Not run)
```

`hdbscan_clustering` *HDBSCAN clustering*

Description

HDBSCAN density clustering

Usage

```
hdbscan_clustering(
  min_pts = 10L,
  method = c("eom", "leaf"),
  knn = c("balltree", "kdtree", "adaptive", "fixed"),
  allow_single_cluster = FALSE
)
```

Arguments

`min_pts`	Minimum cluster size and core-distance order (default 10).
`method`	`"eom"` (excess of mass, default) or `"leaf"`. The leaf method uses `dbscan::hdbscan()` on small corpora only.
`knn`	kNN graph construction strategy for the Boruvka MST step.
	`"balltree"` (default) Ball-tree dual-tree Borůvka. Builds a Ball-tree from the data: bounding hyperspheres prune more effectively than axis-aligned boxes in >=3-D, keeping Borůvka rounds O(n log n) even on data without strong cluster separation. kNN results from the core-distance pass are reused as a warm-up so no extra tree traversal is needed. Fastest for real corpus data; recommended default.
	`"kdtree"` KD-tree dual-tree Borůvka (nanoflann). Same algorithm as `"balltree"` but with axis-aligned bounding boxes. Faster to build; pruning degrades in higher dimensions (>=4-D). Use when comparing against the `"balltree"` or for debugging.
	`"adaptive"` No-tree fallback: builds a kNN graph with `dbscan::kNN()` starting at `k = max(min_pts, 15)` and doubles until the MST is fully connected. Guaranteed connectivity; useful when the tree-based methods miss an isolated island cluster.
	`"fixed"` Like `"adaptive"` but caps `k` at 200. Fastest; occasionally misses island clusters beyond the cap.
`allow_single_cluster`	Logical (default `FALSE`, matching Python `hdbscan`'s default). When `FALSE`, a single cluster that covers all points is never returned: if the EOM excess-of-mass algorithm would select the root cluster, the root's immediate children are activated instead, guaranteeing at least two clusters when sub-structure is present. Set to `TRUE` to allow a single-cluster result (appropriate for data that is genuinely unimodal).

Details

Default clustering model for `fit_bertopic`. Uses a Boruvka minimum spanning tree on the mutual-reachability kNN graph (Rcpp) instead of the naive Prim's algorithm in `dbscan::hdbscan()`, which avoids the O(n^2) memory cost that causes OOM at ~30K+ documents. Memory is O(n x k) throughout, making it practical at 100K+ documents.

Value

An `hdbscan_clustering` model object.

Examples

```
m <- hdbscan_clustering(min_pts = 5L)
m_kd <- hdbscan_clustering(min_pts = 5L, knn = "kdtree")
```

`hierarchical_topics` *Build a hierarchical topic tree from a fitted topic model*

Description

Performs agglomerative clustering on the L2-normalised topic centroids stored in `fit$topic_centroids` using cosine distance. At each merge node the function pools the c-TF-IDF scores of all constituent topics and records the top terms, so every level of the hierarchy is labelled.

Usage

```
hierarchical_topics(fit, method = "ward.D2", top_n_terms = 10L)
```

Arguments

`fit`	A `bertopic_fit` object from `fit_bertopic()`.
`method`	Linkage method passed to `stats::hclust()`. `"ward.D2"` (default) tends to produce compact, balanced clusters. Other good choices: `"average"` (UPGMA), `"complete"`.
`top_n_terms`	Number of top terms to record at each internal node.

Details

The result can be visualised with `visualize_hierarchy()`.

Value

A list of class `hierarchical_topics` with elements:

`hclust` the `hclust` object (for use with base R dendrogram functions if desired)

`merge_df` a data frame with one row per merge, recording `parent_id`, `child_left`, `child_right`, `distance` (cosine), `topics` (comma-separated topic IDs), and `terms`

`topic_ids` integer vector of leaf topic IDs in original order

`method` the linkage method used

Examples

```
## Not run:
  enc <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  fit <- fit_bertopic(enc, docs = abstracts)
  h   <- hierarchical_topics(fit)
  print(h)
  visualize_hierarchy(h, fit = fit)

## End(Not run)
```

---

kmeans_clustering *K-means clustering*

Description

Wraps `stats::kmeans`. Unlike HDBSCAN, k-means assigns every document to a cluster (no noise label `-1`) and requires specifying the number of clusters `k` in advance.

Usage

```
kmeans_clustering(k, nstart = 10L, iter.max = 300L)
```

Arguments

`k`	Number of clusters.
`nstart`	Number of random restarts (default 10).
`iter.max`	Maximum iterations (default 300).

Value

A `kmeans_clustering` model object.

Examples

```
m <- kmeans_clustering(k = 5L)
```

label_topics_llm *Label topics using a large language model*

Description

For each non-noise topic, builds a prompt containing the top c-TF-IDF terms and up to `n_representative_docs` representative documents, sends it to an LLM API, and stores the returned label in `fit$topic_labels`.

Usage

```
label_topics_llm(
  fit,
  provider = c("anthropic", "openai", "ollama"),
  api_key = NULL,
  model = NULL,
  top_n_terms = 10L,
  n_representative_docs = 3L,
  custom_prompt = NULL,
  verbose = TRUE
)
```

Arguments

`fit`	A `bertopic_fit` object from `fit_bertopic`.
`provider`	LLM provider: `"anthropic"` (default), `"openai"`, or `"ollama"` (local, no API key needed).
`api_key`	API key string. Ignored for `"ollama"`. Defaults to the relevant environment variable for cloud providers.
`model`	Model identifier. Defaults to `"claude-haiku-4-5-20251001"` (Anthropic), `"gpt-4o-mini"` (OpenAI), or `"llama3.2"` (Ollama).
`top_n_terms`	Number of top terms included in the prompt (default 10).
`n_representative_docs`	Number of representative documents included in the prompt (default 3).
`custom_prompt`	Optional character string to override the default prompt template. Use `{terms}` and `{docs}` as placeholders.
`verbose`	Print each returned label as it arrives (default `TRUE`).

Details

Three providers are supported:

`"anthropic"` Anthropic Claude via the Messages API. Requires `api_key` or the `ANTHROPIC_API_KEY` environment variable.

`"openai"` OpenAI GPT via the Chat Completions API. Requires `api_key` or `OPENAI_API_KEY`.

`"ollama"` Local Ollama server (OpenAI-compatible endpoint at `http://localhost:11434/v1`). No API key required. Start the server with `ollama serve` and pull a model with `ollama pull llama3.2` before use.

Requires the **httr2** package.

Value

The input `fit` with updated `$topic_labels`.

Examples

```
## Not run:
  enc <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  fit <- fit_bertopic(docs = abstracts, encoder = enc)
  fit <- label_topics_llm(fit, provider = "anthropic",
                          api_key = Sys.getenv("ANTHROPIC_API_KEY"),
                          model = "claude-haiku-4-5-20251001")
  get_topic_info(fit)

## End(Not run)
```

---

load_bertopic *Load a previously saved BERTopic model*

Description

Reads a model directory written by `save_bertopic` and reconstructs the `bertopic_fit` object.

Usage

```
load_bertopic(path)
```

Arguments

`path`	Path to the directory created by `save_bertopic`.

Value

A `bertopic_fit` object.

See Also

`save_bertopic`

Examples

```
## Not run:
  enc  <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  fit  <- fit_bertopic(docs = abstracts, encoder = enc)
  path <- tempdir()
  save_bertopic(fit, path)
  fit2 <- load_bertopic(path)

## End(Not run)
```

`load_bert_weights` *Load BERT weights from a checkpoint into a constructed model*

Description

Reads weights from a safetensors or PyTorch pickle file, normalizes the parameter names to match the R module structure, filters out task-head keys we don't need (`pooler.*`, `cls.*`, `lm_head.*`, ...), and applies the result via `model$load_state_dict()`.

Usage

```
load_bert_weights(model, weights_path, strict = FALSE)
```

Arguments

`model`	A `bert_model` instance (or compatible nn_module).
`weights_path`	Path to a `.safetensors` or `.bin` file.
`strict`	If TRUE, errors when expected parameters are missing. If FALSE, just warns. Default FALSE – most checkpoints have a handful of extra task-head keys that are correctly ignored.

Details

Most users don't call this directly – it's invoked by `load_hf_bert()`. Exposed for users who construct a model manually or want to load alternative weight files.

Value

Invisibly, the named list of loaded weights.

Examples

```
## Not run:
  enc <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  # load_bert_weights() is called internally by load_hf_bert() and
  # load_specter2(); advanced users can call it directly when injecting
  # custom weights into an existing model object.

## End(Not run)
```

load_cohere_embedder *Load a Cohere embedding model*

Description

Returns an `api_embedder` that calls the Cohere Embed API. Requires the **httr2** package and a valid Cohere API key.

Usage

```
load_cohere_embedder(
  model = "embed-english-v3.0",
  api_key = Sys.getenv("COHERE_API_KEY"),
  input_type = "search_document"
)
```

Arguments

`model`	Cohere embedding model name. • `"embed-english-v3.0"` (default) – 1024-d English • `"embed-multilingual-v3.0"` – 1024-d, 100+ languages • `"embed-english-light-v3.0"` – 384-d, faster/cheaper
`api_key`	API key. Defaults to `Sys.getenv("COHERE_API_KEY")`.
`input_type`	Cohere input type controlling the embedding space used. • `"search_document"` (default) – for indexing documents • `"search_query"` – for embedding queries • `"clustering"` – optimized for clustering • `"classification"` – optimized for classification

Details

The returned object is compatible with `embed_texts()`, `embed_texts_cached()`, and `fit_bertopic()`.

Value

An `api_embedder` object.

Examples

```
## Not run:
  enc <- load_cohere_embedder()   # uses COHERE_API_KEY env var
  emb <- embed_texts(enc, docs)

## End(Not run)
```

load_embeddings *Load an embedding matrix from disk*

Description

Load an embedding matrix from disk

Usage

```
load_embeddings(path)
```

Arguments

path	Path to a `.rds` or `.csv` file previously written by `save_embeddings`.

Value

A numeric matrix.

Examples

```
## Not run:
  enc  <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  emb  <- embed_texts(enc, c("doc one", "doc two"))
  path <- tempfile(fileext = ".rds")
  save_embeddings(emb, path)
  load_embeddings(path)

## End(Not run)
```

load_hf_bert *Load a BERT-family model from HuggingFace for use in R*

Description

Downloads a model's config, weights, and tokenizer from the HuggingFace Hub and returns an "encoder" object that can be passed to `embed_texts()` or `fit_bertopic()`. Works for any model with a standard BERT architecture: vanilla BERT, MiniLM, MPNet, SciBERT, BioBERT, ClinicalBERT, BERT-for-Patents, FinBERT, LegalBERT, the sentence-transformers built on top of these, and so on.

Usage

```
load_hf_bert(repo_id, weights_path = NULL, prefix = "")
```

Arguments

repo_id	A HuggingFace repo ID, e.g. "sentence-transformers/all-MiniLM-L6-v2" or "NetworkIsLife/SciBert_Cased_DAFS".
weights_path	Optional path to a local weights file (.safetensors or .bin). Useful when (a) the upstream model lacks safetensors and you've downloaded a converted version from a PR branch, (b) you've produced a local conversion, or (c) you want to pin to a specific local file rather than the latest on the Hub. When supplied, this overrides the Hub weight discovery; config.json and the tokenizer are still pulled from repo_id.
prefix	Optional string prepended to every text before tokenization. Required for best performance with BGE ("Represent this sentence: ") and E5 ("passage: ") models. Stored in the returned encoder and applied automatically by embed_texts(); pass prefix = "" to disable.

Details

The function tries to use modern formats when available and falls back transparently when they aren't:

- **Weights**: model.safetensors is preferred (device-agnostic, safe, fast). If absent, pytorch_model.bin is read via R-torch's pickle loader. If the model has only pytorch_model.bin and that file was saved on a CUDA device, the load will fail – convert the upstream model to safetensors first (see HuggingFace's safetensors/convert Space) and pass the result via weights_path.
- **Tokenizer**: tokenizer.json (HuggingFace fast tokenizer via the tok package) is preferred. If absent, falls back to make_wordpiece_tokenizer() reading vocab.txt (requires the wordpiece package).

Value

A list with three elements:

model the loaded bert_model nn_module

tokenizer a tokenizer object exposing encode_batch()

config the architecture config as a list

Examples

```
## Not run:
  enc <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  emb <- embed_texts(enc, c("Hello world.", "Another sentence."))

  # BGE model  --  prefix at load time, applied automatically on every embed call
  enc <- load_hf_bert("BAAI/bge-base-en-v1.5",
                      prefix = "Represent this sentence: ")

  # E5 model  --  passage prefix for documents
  enc <- load_hf_bert("intfloat/e5-base-v2", prefix = "passage: ")
  # Override per-call for query-side embedding:
  query_emb <- embed_texts(enc, queries, prefix = "query: ")
```

```
  # With a custom weights file (e.g. safetensors from an unmerged PR)
  enc <- load_hf_bert("pritamdeka/S-Scibert-snli-multinli-stsb",
                      weights_path = "/path/to/local/model.safetensors")

## End(Not run)
```

`load_hf_classifier` *Load a fine-tuned BERT-family classifier from HuggingFace*

Description

Downloads `config.json`, model weights, and a tokenizer from the HuggingFace Hub and returns an `hf_classifier` object. Supports sequence classification (sentiment, topic, intent), multi-label classification, regression (e.g. VAD emotion scores), and token classification (NER, POS tagging).

Usage

```
load_hf_classifier(repo_id, weights_path = NULL, prefix = "")
```

Arguments

`repo_id`	HuggingFace repo ID, e.g. `"cardiffnlp/twitter-xlm-roberta-base-sentiment"` or `"RobroKools/vad-bert"`.
`weights_path`	Optional path to a local weights file (`.safetensors` or `.bin`). Overrides Hub weight discovery; config and tokenizer still come from `repo_id`.
`prefix`	Optional string prepended to every text before tokenization (useful for instruction-tuned classifiers). Default `""`.

Details

The task type and output format are auto-detected from `config.json`:

- `architectures` containing `"ForTokenClassification"` -> NER mode
- `problem_type = "regression"` -> return raw numeric scores
- `problem_type = "multi_label_classification"` -> sigmoid per label
- otherwise -> softmax single-label classification

Supported backbone architectures: the same as `load_hf_bert()` – BERT, RoBERTa, XLM-RoBERTa, CamemBERT, DistilBERT (backbone only), MPNet.

Value

An `hf_classifier` list with elements `model`, `tokenizer`, `config`, `id2label`, `problem_type`, `task`, `num_labels`, `repo_id`, `prefix`.

Examples

```
## Not run:
  # Sentiment analysis (single-label, 3 classes)
  clf <- load_hf_classifier("cardiffnlp/twitter-xlm-roberta-base-sentiment")
  classify_texts(clf, c("I love this!", "Terrible experience."))

  # VAD regression (valence, arousal, dominance)
  clf <- load_hf_classifier("RobroKools/vad-bert")
  classify_texts(clf, c("I am ecstatic!", "Feeling nervous about the exam."))

  # Named entity recognition
  clf <- load_hf_classifier("dslim/bert-base-NER")
  classify_texts(clf, c("Albert Einstein was born in Ulm, Germany."))

## End(Not run)
```

---

load_openai_embedder *Load an OpenAI embedding model*

Description

Returns an `api_embedder` that calls the OpenAI Embeddings API. Requires the **httr2** package and a valid OpenAI API key.

Usage

```
load_openai_embedder(
  model = "text-embedding-3-small",
  api_key = Sys.getenv("OPENAI_API_KEY"),
  dimensions = NULL,
  base_url = "https://api.openai.com/v1"
)
```

Arguments

`model`	OpenAI embedding model name. • `"text-embedding-3-small"` (default) – 1536-d, fast and cheap • `"text-embedding-3-large"` – 3072-d, highest quality • `"text-embedding-ada-002"` – legacy 1536-d model
`api_key`	API key. Defaults to `Sys.getenv("OPENAI_API_KEY")`.
`dimensions`	Optional integer to request a reduced output dimension (only supported by `text-embedding-3-*` models).
`base_url`	API base URL. Override for Azure OpenAI or API proxies.

Details

The returned object is compatible with `embed_texts()`, `embed_texts_cached()`, and `fit_bertopic()`.

Value

An `api_embedder` object.

Examples

```
## Not run:
  enc <- load_openai_embedder()   # uses OPENAI_API_KEY env var
  emb <- embed_texts(enc, c("Transformers in R.", "Topic modelling."))
  fit <- fit_bertopic(enc, docs = abstracts)

## End(Not run)
```

`load_specter2` *Load a SPECTER2 model with a task-specific adapter*

Description

SPECTER2 doi:10.48550/arXiv.2211.13308 extends the original SPECTER model with task-specific Pfeiffer adapters trained on millions of citation pairs. This function loads the base encoder from `allenai/specter2_base` and injects the chosen adapter into each transformer layer, matching the behaviour of the Python `adapters` library.

Usage

```
load_specter2(
  adapter = "NetworkIsLife/specter2",
  base = "NetworkIsLife/specter2_base",
  adapter_name = "[PRX]"
)
```

Arguments

`adapter`	HuggingFace repo ID of the adapter checkpoint. Must contain a `pytorch_adapter.bin` file and an `adapter_config.json`. Default: `"allenai/specter2"` (proximity adapter).
`base`	HuggingFace repo ID of the base model. Default: `"allenai/specter2_base"`.
`adapter_name`	Name of the adapter as stored in the checkpoint's weight keys. For all official SPECTER2 adapters this is `"[PRX]"`.

Details

Available adapters on the HuggingFace Hub:

`"allenai/specter2"` Proximity / similarity – recommended for document retrieval and topic modeling (default).

`"allenai/specter2_adhoc_query"` Query-side adapter for asymmetric retrieval (query vs. document).

`"allenai/specter2_classification"` Trained for paper classification tasks.

The returned object is a standard `bert_encoder` compatible with `embed_texts()`, `embed_texts_cached()`, and `fit_bertopic()`.

Value

A `bert_encoder` object (same class as `load_hf_bert()`) with Pfeiffer adapter modules injected into every transformer layer. The list also carries `$adapter_repo` recording which adapter was loaded.

Examples

```
## Not run:
  enc <- load_specter2()   # proximity adapter  --  best for topic modeling
  emb <- embed_texts(enc, c("Graph neural networks for drug discovery.",
                            "Climate tipping points and carbon budgets."))

## End(Not run)
```

---

load_stopwords *Load a stopword list from a character vector, data frame, or file*

Description

Accepts three input types, making it easy to manage domain-specific stopwords from any source:

Character vector Returned as-is (after deduplication and trimming).

Data frame Words are taken from `column` (or the first column when `column` is `NULL`).

File path Supports two file types:

- **Plain text** (`.txt` or no extension): one word per line, blank lines are ignored.
- **Tabular** (`.csv` or `.tsv`): read as a data frame and words extracted from `column` (or the first column).

Usage

```
load_stopwords(source, column = NULL)
```

Arguments

`source`	A character vector of words, a file path, or a data frame.
`column`	Name of the column to use when `source` is a data frame or tabular file. Defaults to the first column.

Details

The result is typically combined with the built-in list before passing to `fit_bertopic`:

```
domain_words <- load_stopwords("domain_stop.txt")
fit <- fit_bertopic(enc, docs,
                    extra_stopwords = domain_words)
```

Value

A deduplicated character vector of stopwords (lowercased and whitespace-trimmed).

Examples

```
load_stopwords(c("foo", "bar", "baz"))
load_stopwords(data.frame(word = c("foo", "bar")), column = "word")
```

`make_wordpiece_tokenizer` *Create a WordPiece tokenizer for BERT models that lack* `tokenizer.json`

Description

Older BERT-family models (SciBERT, BioBERT, the original BERT-base, etc.) ship a `vocab.txt` file but not the HuggingFace fast-tokenizer format. This function builds a tokenizer object exposing the same methods as `tok::tokenizer$from_pretrained()` (`encode_batch`, `enable_padding`, `enable_truncation`) so it can be used interchangeably with `embed_texts()`.

Usage

```
make_wordpiece_tokenizer(vocab_path, do_lower_case = NULL, max_length = 512L)
```

Arguments

`vocab_path`	Path to the model's `vocab.txt` file.
`do_lower_case`	Logical, whether to lowercase input before tokenization. If `NULL` (the default), inferred from the vocabulary: if no tokens start with an uppercase letter, the vocab is treated as uncased. Pass an explicit value if `tokenizer_config.json` specifies it.
`max_length`	Default maximum sequence length, including the two special tokens `[CLS]` and `[SEP]`. Can be overridden later via `tokenizer$enable_truncation()`.

Details

Uses the CRAN package `wordpiece` for the WordPiece algorithm itself.

Value

A list with methods `encode_batch(texts)`, `enable_padding()`, and `enable_truncation(max_length)`.

Examples

```
## Not run:
  vocab <- hfhub::hub_download("allenai/scibert_scivocab_cased", "vocab.txt")
  tk <- make_wordpiece_tokenizer(vocab)
  tk$encode_batch(c("This is a sentence.", "Another one."))

## End(Not run)
```

`mean_pool` *Mean-pool token-level hidden states into a sentence vector*

Description

Takes the encoder's final hidden states `(B, L, H)` and the attention mask `(B, L)`, masks out padding positions, and averages along the sequence dimension to produce `(B, H)` sentence vectors.

Usage

```
mean_pool(hidden, attention_mask)
```

Arguments

`hidden`	A 3-D `torch_tensor` of shape `(batch, seq_len, hidden)`.
`attention_mask`	An integer `torch_tensor` of shape `(batch, seq_len)` with 1 for real tokens and 0 for padding.

Value

A 2-D `torch_tensor` of shape `(batch, hidden)`.

Examples

```
## Not run:
  enc    <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  tokens <- enc$tokenizer$encode_batch(c("hello world"))
  ids    <- torch::torch_tensor(matrix(tokens[[1L]]$ids, nrow = 1L),
                                dtype = torch::torch_long())
  mask   <- torch::torch_tensor(matrix(tokens[[1L]]$attention_mask, nrow = 1L),
                                dtype = torch::torch_long())
  hidden <- enc$model(ids, mask)
  mean_pool(hidden, mask)

## End(Not run)
```

merge_topics *Manually merge a set of topics into one*

Description

All topics in `topics_to_merge` are combined into the one with the smallest ID. All derived state (c-TF-IDF terms, labels, centroids, representative docs) is recomputed from the merged assignments.

Usage

```
merge_topics(fit, topics_to_merge)
```

Arguments

`fit`	A `bertopic_fit` object from `fit_bertopic`.
`topics_to_merge`	Integer vector of at least two topic IDs to merge.

Value

An updated `bertopic_fit`.

Examples

```
## Not run:
  enc <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  fit <- fit_bertopic(docs = abstracts, encoder = enc)
  fit <- merge_topics(fit, topics_to_merge = c(0L, 1L))

## End(Not run)
```

no_reduction *Skip dimensionality reduction (identity pass-through)*

Description

Passes the raw embeddings directly to the clustering step. Useful when the embeddings are already low-dimensional or when you want to cluster in the original space.

Usage

```
no_reduction()
```

Value

A `no_reduction` model object.

Examples

```
m <- no_reduction()
```

`pca_reduction` *PCA dimensionality reduction*

Description

Wraps `stats::prcomp`. Unlike UMAP, PCA is deterministic and fast, but often produces lower-quality cluster separation for text embeddings.

Usage

```
pca_reduction(n_components = 5L, scale. = FALSE)
```

Arguments

`n_components`	Number of principal components to retain.
`scale.`	Whether to scale variables before computing PCA (default `FALSE`; BERT embeddings are already normalised).

Value

A `pca_reduction` model object.

Examples

```
m <- pca_reduction(n_components = 3L)
```

`pos_dtm` *Build a POS-filtered document-term matrix*

Description

Annotates each document in `docs` with a udpipe language model, keeps only tokens whose UPOS tag is in `pos` (and/or phrases matching `patterns`), then builds a sparse document-term matrix from the surviving tokens. The result is compatible with `c_tf_idf`.

Usage

```
pos_dtm(
  docs,
  pos = c("NOUN", "PROPN"),
  patterns = NULL,
  lemmatize = TRUE,
  language = "english",
  model_dir = NULL,
  min_df = 2L,
  max_df_frac = 0.95,
  extra_stopwords = character(0L),
  verbose = TRUE
)
```

Arguments

docs	Character vector of documents.
pos	UPOS tags to retain. Default c("NOUN","PROPN").
patterns	Optional list of UPOS sequences for multi-word phrases.
lemmatize	Use lemmatised forms (default TRUE).
language	udpipe language name (default "english").
model_dir	Directory for the cached language model.
min_df	Minimum document frequency (default 2).
max_df_frac	Maximum document-frequency fraction (default 0.95).
extra_stopwords	Character vector of additional words to exclude.
verbose	Print progress messages (default TRUE).

Value

A sparse dgCMatrix: documents x POS-filtered vocabulary.

See Also

pos_representation, build_dtm

Examples

```
## Not run:
  docs <- c("The neural network learns features.",
            "Gradient descent optimizes weights.",
            "Transformers use attention mechanisms.")
  pos_dtm(docs, pos = c("NOUN", "VERB"))

## End(Not run)
```

pos_representation *Construct a POS-based representation model*

Description

Use as the `representation_model` argument in `fit_bertopic` to restrict the c-TF-IDF vocabulary to tokens whose Universal POS (UPOS) tag is in `pos`, or to phrases that match specific POS-tag sequences.

Usage

```
pos_representation(
  pos = c("NOUN", "PROPN"),
  patterns = NULL,
  lemmatize = TRUE,
  language = "english",
  model_dir = NULL,
  min_df = 2L,
  max_df_frac = 0.95
)
```

Arguments

`pos`	Character vector of UPOS tags to retain as single tokens. Default `c("NOUN","PROPN")`. Set to `NULL` to skip single-token filtering and rely on `patterns` alone.
`patterns`	Optional list of UPOS sequences to extract as multi-word phrases, e.g. `list(c("ADJ","NOUN"), c("NOUN","NOUN"))`. Matched consecutive tokens are joined with `"_"` and added to the vocabulary.
`lemmatize`	Use lemmatised forms rather than surface tokens (default `TRUE`).
`language`	Language name understood by udpipe, e.g. `"english"`, `"dutch"`, `"french"`. Default `"english"`.
`model_dir`	Directory to cache the udpipe language model. Defaults to a session-scoped temporary directory.
`min_df`	Minimum document frequency for a term to survive (default 2).
`max_df_frac`	Maximum document-frequency fraction (default 0.95).

Details

Requires the **udpipe** package (`install.packages("udpipe")`). A language model (~15 MB) is downloaded from the Universal Dependencies collection on first use and cached in `model_dir`.

Common UPOS tags:

`NOUN` Common nouns (default, with `PROPN`)

`PROPN` Proper nouns

`VERB` Main verbs – use for action-focused topics

ADJ Adjectives

ADV Adverbs

Value

An object of class pos_representation.

See Also

pos_dtm, fit_bertopic, cvalue_representation

Examples

```
m <- pos_representation(pos = c("NOUN", "PROPN", "ADJ"))
```

---

predict.bertopic_fit *Predict topics for new documents using a fitted BERTopic model*

Description

Embeds new documents (or accepts pre-computed embeddings) and assigns each to the nearest topic centroid by cosine similarity. Noise (-1) is never assigned as a target – all new documents receive a real topic.

Usage

```
## S3 method for class 'bertopic_fit'
predict(object, new_docs, encoder = NULL, embeddings = NULL, ...)
```

Arguments

object	A bertopic_fit from fit_bertopic.
new_docs	Character vector of documents to predict.
encoder	Optional encoder from load_hf_bert. Required when embeddings is NULL.
embeddings	Optional numeric matrix of pre-computed embeddings (nrow = length(new_docs)). Overrides encoder.
...	Unused (for S3 generic compatibility).

Value

A list with:

topics Integer vector of assigned topic IDs.

probabilities Cosine similarity to the assigned centroid (proxy for confidence, in [0, 1] for normalised embeddings).

all_similarities Numeric matrix (nrow = length(new_docs), ncol = n_topics) of cosine similarities to every topic centroid.

Examples

```
## Not run:
  enc  <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  fit  <- fit_bertopic(docs = abstracts, encoder = enc)
  pred <- predict(fit, new_docs = c("new paper about deep learning"),
                  encoder = enc)
  pred$topics

## End(Not run)
```

`print.bertopic_fit` *Print method for bertopic_fit objects*

Description

Print method for bertopic_fit objects

Usage

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

Arguments

`x`	A bertopic_fit object.
`...`	Unused.

Value

Invisibly returns `x`.

Examples

```
## Not run:
  enc <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  fit <- fit_bertopic(docs = abstracts, encoder = enc)
  print(fit)

## End(Not run)
```

print.bertopic_flow *Print method for bertopic_flow objects*

Description

Print method for bertopic_flow objects

Usage

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

Arguments

x	A bertopic_flow object.
...	Unused.

Value

Invisibly returns x.

Examples

```
## Not run:
  enc  <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  flow <- fit_topics_over_time(enc, docs = abstracts,
                               period_var = rep(c("A", "B"), each = 50L))
  print(flow)

## End(Not run)
```

print.bert_encoder *Print method for bert_encoder objects*

Description

Print method for bert_encoder objects

Usage

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

Arguments

x	A bert_encoder object.
...	Unused.

Value

Invisibly returns `x`.

Examples

```
## Not run:
  enc <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  print(enc)

## End(Not run)
```

`print.hf_classifier` *Print method for hf_classifier objects*

Description

Print method for hf_classifier objects

Usage

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

Arguments

`x`	An hf_classifier object.
`...`	Unused.

Value

Invisibly returns `x`.

Examples

```
## Not run:
  cls <- load_hf_classifier("cardiffnlp/twitter-roberta-base-sentiment-latest")
  print(cls)

## End(Not run)
```

print_topics *Pretty-print discovered topics*

Description

Pretty-print discovered topics

Usage

```
print_topics(fit, max_topics = 20)
```

Arguments

fit	A fit object from fit_bertopic().
max_topics	Maximum number of topics to display.

Value

Called for its side effect (printing); returns NULL invisibly.

Examples

```
## Not run:
  enc <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  fit <- fit_bertopic(docs = abstracts, encoder = enc)
  print_topics(fit)

## End(Not run)
```

reduce_outliers *Reassign noise documents to the nearest real topic*

Description

Documents assigned to the noise cluster (-1) by HDBSCAN are reassigned to the most similar non-noise topic. Only documents whose best similarity exceeds threshold are reassigned; the rest remain as noise.

Usage

```
reduce_outliers(fit, strategy = "embeddings", threshold = 0, verbose = TRUE)
```

Arguments

fit	A bertopic_fit object from fit_bertopic.
strategy	How to measure similarity: "embeddings" **(default)** Cosine similarity between the document embedding and each topic centroid. "c-tf-idf" Cosine similarity between the document's term-frequency vector and each topic's c-TF-IDF vector. Useful when embedding quality is low or unavailable.
threshold	Minimum similarity required to reassign a document. Documents below the threshold remain as noise (-1). Default 0.0 reassigns all noise documents.
verbose	Print a reassignment summary (default TRUE).

Value

An updated bertopic_fit.

Examples

```
## Not run:
  enc <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  fit <- fit_bertopic(docs = abstracts, encoder = enc)
  fit <- reduce_outliers(fit, threshold = 0.1)

## End(Not run)
```

---

reduce_topics *Reduce the number of topics by iteratively merging the most similar pair*

Description

At each step the two non-noise topics whose centroids have the highest cosine similarity are merged. The loop continues until the desired number of topics remains. After the loop, topics are renumbered 0, 1, 2, ... in order of their (post-merge) IDs and all derived state is recomputed.

Usage

```
reduce_topics(fit, nr_topics, verbose = TRUE)
```

Arguments

fit	A bertopic_fit object from fit_bertopic.
nr_topics	Target number of non-noise topics.
verbose	Print progress messages (default TRUE).

Value

An updated `bertopic_fit` with renumbered topics.

Examples

```
## Not run:
  enc <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  fit <- fit_bertopic(docs = abstracts, encoder = enc)
  fit <- reduce_topics(fit, nr_topics = 5L)

## End(Not run)
```

`rhobots_demo` *Run a quick Rhobots demo using classic novels from Project Gutenberg*

Description

Downloads a selection of classic novels, splits them into paragraphs, and runs the full BERTopic pipeline: embed -> UMAP -> HDBSCAN -> c-TF-IDF. Prints a topic summary and shows two interactive plots: a 2-D topic map and a bar chart of top terms per topic.

Usage

```
rhobots_demo(n_per_book = 150L, device = "cpu", seed = 42L, verbose = TRUE)
```

Arguments

`n_per_book`	Maximum paragraphs to sample per book (default 150). Lower values are faster; higher values give richer, more stable topics.
`device`	Device for the encoder: `"cpu"` (default), `"mps"` (Apple Silicon), or `"cuda"` (NVIDIA GPU).
`seed`	Integer random seed for UMAP and sampling (default 42).
`verbose`	Print progress messages (default `TRUE`).

Details

The encoder used is `sentence-transformers/all-MiniLM-L6-v2` (~22 MB), which is downloaded once and cached by **hfhub**. Subsequent runs skip the download.

Value

Invisibly, a list with elements `fit`, `embeddings`, `encoder`, and `texts` (a data frame with columns `text` and `title`), so you can continue exploring after the demo.

Examples

```
## Not run:
  result <- rhobots_demo(n_per_book = 50L)
  print_topics(result$fit)

## End(Not run)
```

---

rhobots_install *Check Rhobots system dependencies and print setup instructions*

Description

Checks whether the 'torch' C++ backend is installed and prints the appropriate setup instructions. Rhobots requires the 'torch' backend (libtorch + lantern, ~560 MB) to run transformer models. Call this function after installing the package to find out what still needs to be done.

Usage

```
rhobots_install()
```

Value

Invisible `NULL`, called for its side-effect of printing instructions.

Examples

```
rhobots_install()
```

---

save_bertopic *Save a fitted BERTopic model to disk*

Description

Writes a `bertopic_fit` object to a directory. The fit is split into:

- `metadata.json` – human-readable summary (topic labels, counts, parameters).
- `fit.rds` – the full fit object minus the large matrices.
- `embeddings.rds` – the document embedding matrix (optional).
- `dtm.rds` – the sparse document-term matrix (optional).

Splitting the heavy matrices means the directory can be browsed and the metadata inspected without loading gigabytes into R.

Usage

```
save_bertopic(
  fit,
  path,
  include_embeddings = TRUE,
  include_dtm = TRUE,
  compress = TRUE
)
```

Arguments

fit	A bertopic_fit object from fit_bertopic.
path	Directory path. Created if it does not exist.
include_embeddings	Save the embedding matrix (default TRUE). Set to FALSE to save space; topic_quality() and reduce_outliers() will not work after reloading.
include_dtm	Save the document-term matrix (default TRUE). Set to FALSE to save space; c-TF-IDF recomputation will not work.
compress	Compress .rds files (default TRUE).

Value

Invisibly, the normalised path.

See Also

load_bertopic

Examples

```
## Not run:
  enc <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  fit <- fit_bertopic(docs = abstracts, encoder = enc)
  save_bertopic(fit, tempdir())

## End(Not run)
```

---

save_embeddings *Save an embedding matrix to disk*

Description

Save an embedding matrix to disk

Usage

```
save_embeddings(embeddings, path)
```

Arguments

embeddings	A numeric matrix (rows = documents, columns = dimensions).
path	File path. Use a .rds extension (recommended – lossless, fast) or .csv (portable but larger and slower). If no extension is given, .rds is appended automatically.

Value

The resolved file path, invisibly.

Examples

```
## Not run:
  enc <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  emb <- embed_texts(enc, c("doc one", "doc two"))
  save_embeddings(emb, tempfile(fileext = ".rds"))

## End(Not run)
```

---

stability_analysis *Measure topic stability across multiple random seeds*

Description

Runs fit_bertopic n_runs times with different random seeds on the same corpus, then measures how consistent the document assignments are across runs using the Adjusted Rand Index (ARI).

Usage

```
stability_analysis(
  docs,
  embeddings,
  n_runs = 5L,
  seeds = NULL,
  ...,
  verbose = TRUE
)
```

Arguments

docs	Character vector of documents.
embeddings	Pre-computed embedding matrix (strongly recommended to avoid re-embedding for every run).
n_runs	Number of independent fits (default 5).
seeds	Integer vector of random seeds, length n_runs. Defaults to 42 * 1:n_runs.
...	Additional arguments forwarded to fit_bertopic.
verbose	Print per-run progress (default TRUE).

Details

ARI = 1 means two clusterings are identical; ARI ≈ 0 means agreement no better than chance; negative values indicate systematic disagreement. A mean ARI above 0.8 across all run pairs indicates a stable, reproducible topic structure.

Value

A list of class `stability_result` with elements:

`ari_matrix` Symmetric ARI matrix (`n_runs` x `n_runs`).

`mean_ari` Mean ARI across all off-diagonal pairs.

`per_doc_stability` For each document, the fraction of runs that agreed on the modal topic assignment (1.0 = always the same).

`n_topics_per_run` Integer vector of non-noise topic counts.

`fits` List of `bertopic_fit` objects (one per run).

See Also

`visualize_stability`, `sweep_topics`

Examples

```
## Not run:
  enc  <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  emb  <- embed_texts(enc, abstracts)
  stab <- stability_analysis(abstracts, emb, n_runs = 3L)
  stab$mean_ari

## End(Not run)
```

---

sweep_topics *Sweep BERTopic hyperparameters and compare topic quality*

Description

Runs `fit_bertopic` and `topic_quality` over a full factorial grid of UMAP and HDBSCAN parameters (and optionally multiple embedding models), then returns a tidy data frame of quality metrics for every combination. Use `visualize_sweep` to compare runs visually.

Usage

```
sweep_topics(
  docs,
  encoders = NULL,
  embeddings = NULL,
  n_neighbors = c(5L, 15L, 30L),
  n_components = c(5L, 10L),
```

```
  min_pts = c(5L, 10L, 20L),
  min_topics = NULL,
  ngram_range = c(1L, 1L),
  top_n_terms = 10L,
  extra_stopwords = NULL,
  sample_size = NULL,
  quality_top_n = 10L,
  quality_sample = 500L,
  quality_space = "reduced",
  seed = 42L,
  verbose = TRUE
)
```

Arguments

docs	Character vector of documents.
encoders	A single encoder (from load_hf_bert) or a named list of encoders. Ignored for any model whose embeddings are supplied via embeddings.
embeddings	A pre-computed numeric matrix (rows = documents, columns = embedding dimensions), or a named list of such matrices when comparing multiple embedding models. Names must match those in encoders when both are supplied.
n_neighbors	Integer vector of UMAP n_neighbors values to try.
n_components	Integer vector of UMAP n_components values to try.
min_pts	Integer vector of HDBSCAN min_pts values to try.
min_topics	Minimum number of topics required in the best selection. When set, best is the highest-silhouette run among those that produced at least min_topics topics. If no combination meets the constraint, a warning is issued and best falls back to the run with the most topics. NULL (default) selects purely by silhouette.
ngram_range, top_n_terms, extra_stopwords	Fixed model parameters passed to fit_bertopic for every run.
sample_size	If not NULL, draw a random sample of this many documents before sweeping. Useful for fast exploration on large corpora.
quality_top_n	Passed to topic_quality as top_n.
quality_sample	Passed to topic_quality as sample_size for the silhouette computation. Default 500.
quality_space	Embedding space for quality metrics; passed to topic_quality as space. Defaults to "reduced", which measures cohesion, separation, and silhouette in the UMAP-reduced space — the space where different n_components and n_neighbors choices produce different cluster structures. Set to "original" to match standalone topic_quality() behaviour (full encoder embedding), but be aware that original-space metrics barely vary across sweep runs for well-separated corpora.
seed	Random seed for reproducibility.
verbose	Print one-line progress per combination.

Details

Embeddings are the expensive step. When sweeping only UMAP/HDBSCAN parameters with a single encoder, pass pre-computed `embeddings` so the encoder runs only once:

```
emb  <- embed_texts(enc, docs, normalize = TRUE)
sw   <- sweep_topics(docs, embeddings = emb, min_pts = c(5, 10, 20))
```

To compare multiple encoders, pass a named list of either encoders or pre-computed matrices:

```
sw <- sweep_topics(docs,
  embeddings = list(minilm = emb1, scibert = emb2),
  n_neighbors = c(5, 15), min_pts = c(5, 10))
```

Value

A list of class `topic_sweep` with elements:

`results` Data frame with one row per parameter combination and columns for all swept parameters plus quality metrics (including `n_topics`).

`best` The selected row of `results`: highest silhouette among runs satisfying `min_topics`, or the run with the most topics if no run satisfies the constraint.

`min_topics` The `min_topics` argument (or `NULL`).

`best_met_constraint` Logical: `TRUE` when `best` satisfies the `min_topics` constraint (always `TRUE` when `min_topics = NULL`).

`n_docs` Number of documents used (after optional sampling).

`sampled` Logical: whether a random sample was drawn.

`param_names` Character vector of swept parameter names.

Examples

```
## Not run:
  enc <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  emb <- embed_texts(enc, abstracts, normalize = TRUE)
  sw  <- sweep_topics(abstracts, embeddings = emb,
                      n_neighbors = c(5L, 15L), min_pts = c(5L, 10L))
  visualize_sweep(sw)

## End(Not run)
```

topics_over_time *Compute how topic representations change over time*

Description

For each unique timestamp (or time bin), the documents assigned to each topic are collected and a local c-TF-IDF representation is computed. Two optional smoothing passes mirror Python BERTopic's behaviour:

Usage

```
topics_over_time(
  fit,
  timestamps,
  nr_bins = NULL,
  evolution_tuning = TRUE,
  global_tuning = TRUE,
  top_n = NULL
)
```

Arguments

fit	A bertopic_fit from fit_bertopic.
timestamps	A vector of the same length as fit$docs giving each document a time label. Can be numeric, Date, POSIXct, or character.
nr_bins	Optional integer. Bin the timestamps into this many equally spaced intervals (using the per-bin median as the representative label).
evolution_tuning	Smooth representations across adjacent timestamps (default TRUE).
global_tuning	Blend each local representation with the global c-TF-IDF (default TRUE).
top_n	Number of top terms to include per (topic, timestamp) row. Defaults to fit$top_n_terms.

Details

evolution_tuning Averages each timestamp's representation with the previous timestamp (L1-normalised) to smooth abrupt changes.

global_tuning Averages each local representation with the global c-TF-IDF (computed over all documents) so words that are absent from a narrow time window are not completely lost.

Value

A data frame (class topics_over_time) with columns Topic, Words, Frequency, Timestamp, sorted by Timestamp then Topic.

Examples

```
## Not run:
  enc  <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  fit  <- fit_bertopic(docs = abstracts, encoder = enc)
 years <- as.Date(paste0(sample(2015:2023, length(abstracts), replace = TRUE), "-01-01"))
  tot  <- topics_over_time(fit, timestamps = years)
  visualize_topics_over_time(tot, fit = fit)

## End(Not run)
```

`topic_coherence` *Compute lexical coherence for discovered topics*

Description

Measures how often the top terms of each topic co-occur in the same documents. High coherence means the words form a semantically tight cluster – they appear together rather than in isolation.

Usage

```
topic_coherence(fit, top_n = 10L, measure = c("npmi", "cv"))
```

Arguments

`fit`	A `bertopic_fit` object from `fit_bertopic`.
`top_n`	Number of top c-TF-IDF terms per topic to include in pairwise co-occurrence calculations (default 10).
`measure`	Coherence measure: `"npmi"` (default) or `"cv"`.

Details

Two measures are supported:

`"npmi"` Normalised Pointwise Mutual Information, the standard measure in recent topic-model benchmarks. Ranges from `-1` (terms never co-occur) to `1` (terms always appear together). Values above `0.1` are generally considered good; above `0.3` is excellent.

`"cv"` The C_V coherence measure, a log-conditional variant that is slightly more discriminative on small corpora. Higher is better; typical range `-4` to `0`.

Value

A list of class `topic_coherence` with elements:

`per_topic` Named numeric vector of per-topic scores.

`mean` Mean coherence across all non-noise topics.

`measure` Which measure was used.

`top_n` Number of terms used.

See Also

topic_quality

Examples

```
## Not run:
  enc <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  fit <- fit_bertopic(docs = abstracts, encoder = enc)
  topic_coherence(fit)

## End(Not run)
```

---

topic_quality *Evaluate topic quality for a fitted BERTopic model*

Description

Computes four families of metrics that characterise different aspects of topic quality:

Cohesion How tightly documents cluster around their topic centroid (mean cosine similarity of each document to its centroid; higher is better).

Separation How distinct topics are from each other (pairwise cosine similarity between L2-normalised topic centroids; lower mean is better, i.e. topics point in different directions in embedding space).

Overlap How much vocabulary topics share (pairwise Jaccard similarity of the top-`top_n` c-TF-IDF terms; lower mean is better).

Distribution How balanced and noise-free the topic assignments are (normalised size entropy, coefficient of variation, noise ratio).

A **silhouette score** in the full embedding space is also computed. It is the standard cluster-quality measure: values near 1 mean documents sit close to their own centroid and far from the nearest other cluster; values near 0 or below indicate overlapping or mis-assigned topics.

Usage

```
topic_quality(
  fit,
  top_n = 10L,
  sample_size = 2000L,
  space = c("original", "reduced")
)
```

Arguments

fit	A bertopic_fit object from fit_bertopic.
top_n	Number of top c-TF-IDF terms per topic used for the Jaccard overlap computation. Default 10.
sample_size	Maximum number of non-noise documents used when computing the silhouette score (which requires an $n \times n$ distance matrix). Set to NULL for exact computation. Default 2000.
space	Embedding space used for cohesion, separation, and silhouette. "original" (default) uses the full encoder embedding (e.g. 384-D); "reduced" uses the UMAP-reduced embedding stored in fit$reduced. The reduced space reflects the actual input to HDBSCAN and shows more variation across different UMAP/HDBSCAN parameter choices, making it the natural choice when comparing sweep runs.

Value

A list of class topic_quality with elements:

cohesion List with global (scalar) and per_topic (named vector) mean doc-to-centroid cosine similarity.

separation List with mean_inter_topic_similarity (scalar) and centroid_similarity (symmetric matrix).

overlap List with mean_jaccard (scalar) and jaccard_matrix (symmetric matrix of pairwise Jaccard scores).

distribution List with counts (named integer vector), noise_ratio, entropy (normalised, in $[0, 1]$), and cv (coefficient of variation of topic sizes).

silhouette List with global, per_topic, sampled (logical), and sample_n.

Examples

```
## Not run:
  enc <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  fit <- fit_bertopic(docs = abstracts, encoder = enc)
  q   <- topic_quality(fit)
  print(q)

## End(Not run)
```

---

transform_bertopic *Predict topics for new documents (standalone alias)*

Description

Wraps predict.bertopic_fit; see that function for full parameter documentation.

Usage

```
transform_bertopic(fit, new_docs, encoder = NULL, embeddings = NULL)
```

Arguments

fit	A bertopic_fit from fit_bertopic.
new_docs	Character vector of new documents.
encoder	Optional encoder from load_hf_bert.
embeddings	Optional pre-computed embedding matrix.

Value

Same as predict.bertopic_fit.

Examples

```
## Not run:
  enc  <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  fit  <- fit_bertopic(docs = abstracts, encoder = enc)
  pred <- transform_bertopic(fit, new_docs = c("new document"), encoder = enc)
  pred$topics

## End(Not run)
```

---

umap_reduction *UMAP dimensionality reduction*

Description

Wraps uwot::umap. This is the default dimensionality-reduction model used by fit_bertopic.

Usage

```
umap_reduction(
  n_neighbors = 15L,
  n_components = 5L,
  min_dist = 0,
  metric = "cosine"
)
```

Arguments

n_neighbors, n_components, min_dist, metric	Passed directly to uwot::umap.

Value

A umap_reduction model object.

Examples

```
m <- umap_reduction(n_neighbors = 10L, n_components = 3L)
```

`visualize_barchart` *Bar charts of top terms per topic*

Description

Produces a grid of horizontal bar charts (via **plotly**), one panel per topic, showing c-TF-IDF scores for the top `top_n` terms. The best term is always at the top of each panel.

Usage

```
visualize_barchart(
  fit,
  topics = NULL,
  top_n = 8L,
  n_cols = 4L,
  width = NULL,
  height = NULL
)
```

Arguments

`fit`	A `bertopic_fit` object from `fit_bertopic`.
`topics`	Integer vector of topic IDs to include. `NULL` (default) shows all non-noise topics.
`top_n`	Number of terms to show per topic (default 8).
`n_cols`	Number of panel columns in the grid (default 4).
`width, height`	Plot dimensions in pixels. Auto-scaled to the grid size when `NULL`.

Value

A `plotly` figure object.

Examples

```
## Not run:
  enc <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  fit <- fit_bertopic(docs = abstracts, encoder = enc)
  visualize_barchart(fit, top_n = 5L)

## End(Not run)
```

`visualize_comparison` *Interactive heatmap of topic - group associations*

Description

Displays the signed chi-square contributions or $\log_2$ ratios from `compare_topics` as a diverging colour heatmap: blue = over- represented in that group, red = under-represented.

Usage

```
visualize_comparison(
  comp,
  top_n_topics = NULL,
  max_label_chars = 25L,
  width = 1000L,
  height = 420L
)
```

Arguments

`comp`	A `compare_topics_result` from `compare_topics`.
`top_n_topics`	Restrict to the `top_n_topics` topics with the largest mean absolute statistic. `NULL` (default) shows all topics.
`max_label_chars`	Maximum characters for topic labels before truncation with an ellipsis (default 25).
`width, height`	Plot dimensions in pixels (default 1000 x 420).

Details

Topics are placed on the x-axis (with angled labels) and groups on the y-axis, which keeps the chart readable even when topic labels are long.

Value

A `plotly` figure.

Examples

```
## Not run:
  enc    <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  fit    <- fit_bertopic(docs = abstracts, encoder = enc)
  groups <- sample(c("A", "B"), length(abstracts), replace = TRUE)
  comp   <- compare_topics(fit, groups)
  visualize_comparison(comp)

## End(Not run)
```

`visualize_hierarchy` *Visualise the hierarchical topic tree as a dendrogram*

Description

Produces an interactive plotly dendrogram with:

- Leaf labels: topic ID + top c-TF-IDF words (from `fit`)
- Internal node hover: merged topics and their pooled terms
- Height axis: cosine distance at which topics were merged

Usage

```
visualize_hierarchy(
  h,
  fit = NULL,
  n_label_words = 3L,
  width = 900L,
  height = 600L
)
```

Arguments

`h`	A `hierarchical_topics` object from `hierarchical_topics()`.
`fit`	Optional `bertopic_fit` – when supplied, leaf labels show topic words; when `NULL`, leaves are labelled by topic ID only.
`n_label_words`	Number of topic words to show per leaf label.
`width, height`	Plot dimensions in pixels.

Value

A `plotly` figure object.

Examples

```
## Not run:
  enc <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  fit <- fit_bertopic(docs = abstracts, encoder = enc)
  h   <- hierarchical_topics(fit)
  visualize_hierarchy(h, fit = fit)

## End(Not run)
```

`visualize_quality` *Visualise topic quality metrics*

Description

Produces a four-panel interactive dashboard (via **plotly**) from a `topic_quality` object returned by `topic_quality`. The panels are:

1. **Silhouette per topic** – horizontal bars coloured from red (negative) through yellow (zero) to green (positive).
2. **Topic size distribution** – document counts per topic, with the noise class shown separately.
3. **Centroid similarity** – pairwise cosine similarity matrix between topic centroids (lower = more distinct).
4. **Vocabulary overlap** – pairwise Jaccard similarity of the top-N c-TF-IDF term sets (lower = less overlap).

Usage

```
visualize_quality(q, fit = NULL, width = 900L, height = 750L)
```

Arguments

`q`	A `topic_quality` object from `topic_quality`.
`fit`	Optional `bertopic_fit` used to resolve short topic labels. When `NULL` topics are labelled `T0`, `T1`, etc.
`width, height`	Plot dimensions in pixels (defaults: 900 x 750).

Value

A `plotly` figure object.

Examples

```
## Not run:
  enc <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  fit <- fit_bertopic(docs = abstracts, encoder = enc)
  q   <- topic_quality(fit)
  visualize_quality(q, fit = fit)

## End(Not run)
```

visualize_stability *Interactive heatmap of pairwise ARI scores*

Description

Interactive heatmap of pairwise ARI scores

Usage

```
visualize_stability(stab, width = 550L, height = 500L)
```

Arguments

stab	A stability_result from stability_analysis.
width, height	Plot dimensions in pixels (default 550 x 500).

Value

A plotly figure.

Examples

```
## Not run:
  enc  <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  emb  <- embed_texts(enc, abstracts)
  stab <- stability_analysis(abstracts, emb, n_runs = 3L)
  visualize_stability(stab)

## End(Not run)
```

visualize_sweep *Visualise the results of a parameter sweep*

Description

Produces an interactive heatmap (via **plotly**) where each row is one parameter combination and each column is a quality metric. Within every column the values are min-max normalised to $[0, 1]$ so that colours are comparable across metrics (green = best in column, white = worst). Metrics where a lower raw value is better (separation, Jaccard overlap, noise percentage) are inverted before normalisation. Hover text shows the actual raw values.

Usage

```
visualize_sweep(
  sweep,
 metrics = c("silhouette", "cohesion", "separation", "jaccard", "entropy", "noise_pct"),
  width = 900L,
  height = NULL
)
```

Arguments

sweep	A topic_sweep object from sweep_topics.
metrics	Character vector selecting which columns of sweep$results to display. "n_topics" is always prepended regardless of this argument.
width, height	Plot dimensions in pixels. Height auto-scales to the number of runs when NULL.

Details

n_topics is always shown as the first column. When a min_topics constraint was passed to sweep_topics, rows that did not meet it are prefixed with "[x] " in the row labels.

Value

A plotly figure object.

Examples

```
## Not run:
  enc <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  emb <- embed_texts(enc, abstracts, normalize = TRUE)
  sw  <- sweep_topics(abstracts, embeddings = emb,
                      n_neighbors = c(5L, 15L), min_pts = c(5L, 10L))
  visualize_sweep(sw)

## End(Not run)
```

---

visualize_topics *Visualise documents in topic space*

Description

Produces an interactive 2-D scatter plot (via **plotly**) with one point per document, coloured by topic.

Usage

```
visualize_topics(
  fit,
  dims = NULL,
  label_topics = TRUE,
  max_label_chars = 30L,
  point_size = 5L,
  noise_color = "#cccccc",
  width = 900L,
  height = 700L
)
```

Arguments

fit	A bertopic_fit object from fit_bertopic.
dims	Either NULL to use the stored 2-D UMAP layout, or a length-2 integer vector selecting columns from fit$reduced.
label_topics	Annotate topic centroids with short labels (default TRUE).
max_label_chars	Truncate centroid labels to this many characters.
point_size	Marker size (default 5).
noise_color	Hex colour for noise documents (default "#cccccc").
width, height	Plot dimensions in pixels (defaults: 900 x 700).

Value

A plotly figure object, or NULL if no topics were found.

Examples

```
## Not run:
  enc <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  fit <- fit_bertopic(docs = abstracts, encoder = enc)
  visualize_topics(fit)

## End(Not run)
```

---

visualize_topics_over_time

Visualise topic frequency over time

Description

Produces an interactive Plotly line chart showing how the relative frequency of each topic changes across timestamps. Hovering over a point shows the topic's top words at that time.

Usage

```
visualize_topics_over_time(
  tot,
  topics = NULL,
  normalize = TRUE,
  width = 900L,
  height = 550L
)
```

Arguments

tot	A topics_over_time data frame from topics_over_time.
topics	Integer vector of topic IDs to include. NULL (default) shows the 10 most frequent topics overall.
normalize	If TRUE (default), the Y-axis shows each topic's share of documents in that time period. If FALSE, raw document counts are shown.
width, height	Plot dimensions in pixels.

Value

A plotly figure.

Examples

```
## Not run:
  enc  <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  fit  <- fit_bertopic(docs = abstracts, encoder = enc)
  years <- as.Date(paste0(sample(2015:2023, length(abstracts), replace = TRUE), "-01-01"))
  tot  <- topics_over_time(fit, timestamps = years)
  visualize_topics_over_time(tot, fit = fit)

## End(Not run)
```

---

visualize_topic_flow *Sankey diagram of topic flow across periods*

Description

Produces an interactive Plotly Sankey diagram where each column represents one time period, each node is a topic (sized by document count), and each link is a cross-period topic similarity above the fitted threshold. Link thickness scales with similarity x min(count_from, count_to).

Usage

```
visualize_topic_flow(
  flow,
  periods = NULL,
  color_by = c("period", "status"),
  width = 1100L,
  height = 650L
)
```

Arguments

flow	A bertopic_flow object from fit_topics_over_time.
periods	Character vector of period labels to include. NULL (default) shows all periods.
color_by	One of "period" (all topics in the same period share a hue) or "status" (nodes coloured by lifecycle status).
width, height	Plot dimensions in pixels.

Value

A plotly figure.

Examples

```
## Not run:
  enc  <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  flow <- fit_topics_over_time(enc, docs = abstracts,
                               period_var = rep(c("A", "B"), each = 50L))
  visualize_topic_flow(flow)

## End(Not run)
```

---

zero_shot_topics *Zero-shot topic modeling with user-defined topic labels*

Description

Instead of discovering topics through unsupervised clustering, the user supplies a named character vector of topic descriptions. Each description is embedded and every document is assigned to the nearest topic by cosine similarity. Documents whose best similarity falls below threshold are labelled as noise (-1).

Usage

```
zero_shot_topics(
  docs,
  labels,
  embeddings = NULL,
  encoder = NULL,
  label_embeddings = NULL,
  threshold = 0,
  ngram_range = c(1L, 2L),
  top_n_terms = 10L,
  extra_stopwords = character(0L),
  verbose = TRUE
)
```

Arguments

docs	Character vector of documents.
labels	Named character vector of topic descriptions. Names become topic labels (e.g. "Climate policy"); values are the text used to compute the embedding anchor (e.g. "carbon emissions renewable energy net-zero").
embeddings	Optional pre-computed numeric matrix for docs (nrow = length(docs)).
encoder	Optional encoder from load_hf_bert. Required when embeddings is NULL, or to embed the label descriptions.
label_embeddings	Optional pre-computed numeric matrix for the label descriptions (nrow = length(labels)). When both embeddings and label_embeddings are supplied, no encoder is needed.
threshold	Minimum cosine similarity for assignment. Documents below this threshold are assigned to noise (-1). Default 0.0 forces every document into some topic.
ngram_range	Integer vector c(min, max) for n-gram extraction in the c-TF-IDF step (default c(1L, 2L)).
top_n_terms	Number of c-TF-IDF terms stored per topic (default 10).
extra_stopwords	Additional stopwords – character vector, file path, or data frame (same formats accepted by fit_bertopic).
verbose	Print progress messages (default TRUE).

Details

The returned object is a full bertopic_fit compatible with all Rhobots accessor and visualisation functions (get_topic_info(), visualize_barchart(), topic_quality(), etc.).

Value

A bertopic_fit object. The extra field $label_names records the original user-supplied label names.

See Also

fit_bertopic, guided_fit_bertopic

Examples

```
## Not run:
  enc    <- load_hf_bert("sentence-transformers/all-MiniLM-L6-v2")
  labels <- c(
    "Climate change" = "carbon emissions global warming climate",
    "Machine learning" = "neural network deep learning model training"
  )
  fit <- zero_shot_topics(abstracts, labels = labels, encoder = enc)
  get_topic_info(fit)

## End(Not run)
```

Index