Package 'TextAnalysisR'

July 27, 2026

Title A Text Mining Workflow Tool

Version 0.1.4

Description Provides a text mining and natural language processing workflow for documents. Includes preprocessing via 'quanteda', lexical analysis (term frequency-inverse document frequency, log-odds ratios, lexical diversity) via 'tidytext', topic modeling via 'stm' and the 'BERTopic' approach, semantic similarity and document clustering on transformer representations, an interactive 'Shiny' interface with 'ggplot2' visualization, optional 'spaCy' preprocessing, and local 'sentence-transformers' or web-based ('OpenAI', 'Gemini') model providers for retrieval-augmented generation, as described in Shin et al. (2026) <doi:10.1177/07319487251412879>.

License GPL (>= 3)

Language en-US

Encoding UTF-8

LazyData true

LazyDataCompression xz

Imports dplyr, DT, ggplot2, htmltools, htmlwidgets, igraph, magrittr, Matrix, methods, quanteda, quanteda.textstats, RColorBrewer, rlang, scales, shiny, tibble, tidyr, tidytext, widyr, withr

Suggests aricode, tidylo, broom, patchwork, cluster, clusterCrit, colourpicker, covr, dbscan, digest, httr, jsonlite, knitr, markdown, MASS, moments, numform, officer, openxlsx, pdftools, progress, proxy, pscl, purrr, readxl, reticulate (>= 1.28), rmarkdown, roxygen2, Rtsne, shinyBS, shinybusy, shinyjs, spelling, stm, stopwords, stringr, syuzhet, testthat (>= 3.0.0), textdata, umap, plotly

Depends R (>= 4.0)

SystemRequirements Python (>= 3.8) with spaCy (>= 3.5) for lemmatization and sentence-transformers (>= 2.2) for embeddings (optional)

URL https://mshin77.github.io/TextAnalysisR/,
https://github.com/mshin77/TextAnalysisR

BugReports https://github.com/mshin77/TextAnalysisR/issues

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Mikyung Shin [aut, cre] (ORCID:
<https://orcid.org/0000-0001-7907-9193>)

Maintainer Mikyung Shin <shin.mikyung@gmail.com>

Repository CRAN

Date/Publication 2026-07-27 14:40:13 UTC

Contents

acronym *Acronym List*

Description

A dataset containing common acronyms used in text processing

Format

A character vector of acronyms

analyze_document_clustering *Analyze Document Clustering*

Description

Complete document clustering analysis with dimensionality reduction and optional clustering

Usage

```
analyze_document_clustering(
  feature_matrix,
  method = "UMAP",
  clustering_method = "none",
  ...
)
```

Arguments

feature_matrix	Feature matrix (documents x features)
method	Dimensionality reduction method ("PCA", "t-SNE", "UMAP")
clustering_method	Clustering method ("none", "kmeans", "hierarchical", "dbscan", "hdbscan")
...	Additional parameters for methods

Value

List containing coordinates, clusters, method info, and quality metrics

analyze_semantic_evolution *Analyze Semantic Evolution*

Description

Analyzes semantic evolution patterns in temporal results

Usage

```
analyze_semantic_evolution(temporal_results, verbose = FALSE, ...)
```

Arguments

temporal_results	Temporal analysis results
verbose	Logical indicating whether to print progress messages
...	Additional parameters

Value

List containing evolution analysis

analyze_sentiment *Analyze Text Sentiment*

Description

Performs sentiment analysis on text data using the syuzhet package. Classification uses the per-token average score with a +/- 0.05 neutral band, so document length does not drive polarity. Lexicon methods score words in isolation; negation, irony, and idioms are not handled. For negation-sensitive text, see analyze_sentiment_llm().

Usage

```
analyze_sentiment(texts, method = "syuzhet", doc_ids = NULL)
```

Arguments

texts	Character vector of texts to analyze
method	Sentiment analysis method: "syuzhet", "bing", "afinn", or "nrc" (default: "syuzhet"). All methods score each matched dictionary word once per document regardless of repetition.
doc_ids	Optional character vector of document identifiers (default: NULL)

Value

A data frame with columns:

document Document identifier

text Original text

sentiment_score Raw summed sentiment score (length-dependent)

n_tokens Number of whitespace-delimited tokens

avg_sentiment sentiment_score / n_tokens

sentiment Classification: "positive", "negative", or "neutral"

Examples

```
abstracts <- TextAnalysisR::SpecialEduTech$abstract[1:10]
sentiment_results <- analyze_sentiment(abstracts)
print(sentiment_results)
```

---

analyze_sentiment_llm *LLM-based Sentiment Analysis*

Description

Analyzes sentiment using Large Language Models (OpenAI or Gemini). Provides nuanced sentiment understanding including sarcasm detection, mixed emotions, and contextual interpretation that lexicon-based methods miss.

Requests use temperature 0, which makes outputs near-deterministic but not fully reproducible across runs or model versions; record the returned `model_used` field when reporting results.

Usage

```
analyze_sentiment_llm(
  texts,
  doc_names = NULL,
  provider = c("openai", "gemini"),
  model = NULL,
  api_key = NULL,
  batch_size = 5,
  include_explanation = FALSE,
  verbose = TRUE
)
```

Arguments

texts	Character vector of texts to analyze.
doc_names	Optional character vector of document names (default: text1, text2, ...).
provider	AI provider to use: "openai" (default) or "gemini".
model	Model name. If NULL, uses provider defaults: "gpt-4.1-mini" (OpenAI), "gemini-2.5-flash-lite" (Gemini).
api_key	API key for OpenAI or Gemini. If NULL, uses environment variable.
batch_size	Number of texts to process per API call (default: 5). Larger batches are more efficient but may hit token limits.
include_explanation	Logical, if TRUE includes natural language explanation for each sentiment classification (default: FALSE).
verbose	Logical, if TRUE prints progress messages (default: TRUE).

Details

LLM-based sentiment analysis offers several advantages over lexicon methods:

- Understands context and nuance
- Detects sarcasm and irony
- Handles mixed emotions
- Works across domains without retraining

Value

A list containing:

document_sentiment Data frame with document-level sentiment scores

summary_stats Summary statistics of the analysis

model_used Model name used for analysis

provider AI provider used

Examples

```
if (interactive()) {
  abstracts <- TextAnalysisR::SpecialEduTech$abstract[1:5]

  sentiment_openai <- analyze_sentiment_llm(abstracts, provider = "openai")

  sentiment_gemini <- analyze_sentiment_llm(abstracts, provider = "gemini",
                                            include_explanation = TRUE)
}
```

`analyze_similarity_gaps` *Analyze Similarity Gaps Between Categories*

Description

Identifies unique items, missing content, and cross-category learning opportunities based on similarity thresholds. Useful for gap analysis in policy documents, topic comparisons, or any cross-category similarity study.

Usage

```
analyze_similarity_gaps(
  similarity_data,
  ref_var = "ref_id",
  other_var = "other_id",
  similarity_var = "similarity",
  category_var = "other_category",
  ref_label_var = NULL,
  other_label_var = NULL,
  unique_threshold = 0.6,
  cross_policy_min = 0.6,
  cross_policy_max = 0.8
)
```

Arguments

`similarity_data`
: A data frame with cross-category similarities, containing:
ref_var Reference item identifier
other_var Comparison item identifier
similarity_var Similarity score
category_var Category of comparison item

`ref_var` Name of column with reference item IDs (default: "ref_id").

`other_var` Name of column with comparison item IDs (default: "other_id").

`similarity_var` Name of column with similarity values (default: "similarity").

`category_var` Name of column with category information (default: "other_category").

`ref_label_var` Optional column with reference item labels (for output).

`other_label_var`
: Optional column with comparison item labels (for output).

`unique_threshold`
: Threshold below which reference items are considered unique (default: 0.6).

`cross_policy_min`
: Minimum similarity for cross-policy opportunities (default: 0.6).

`cross_policy_max`
: Maximum similarity for cross-policy opportunities (default: 0.8).

Value

A list containing:

`unique_items`	Data frame of reference items with low similarity (unique content)
`missing_items`	Data frame of comparison items with low similarity (content gaps)
`cross_policy`	Data frame of items with moderate similarity (learning opportunities)
`summary_stats`	Summary statistics by category

Examples

```
articles <- TextAnalysisR::SpecialEduTech[1:6, ]
articles$display_name <- paste0("d", seq_len(nrow(articles)))
term_matrix <- as.matrix(quanteda::dfm(quanteda::tokens(articles$abstract)))
normalized_matrix <- term_matrix / sqrt(rowSums(term_matrix ^ 2))
similarity_matrix <- normalized_matrix %*% t(normalized_matrix)
dimnames(similarity_matrix) <- list(articles$display_name, articles$display_name)
cross_similarities <- extract_cross_category_similarities(
  similarity_matrix  = similarity_matrix,
  docs_data          = articles,
  reference_category = "thesis",
  compare_categories = "journal_article",
  category_var       = "reference_type",
  id_var             = "display_name"
)
gap_analysis <- analyze_similarity_gaps(
  similarity_data = cross_similarities,
  ref_var = "ref_id",
  other_var = "other_id",
  similarity_var = "similarity",
  category_var = "other_category",
  unique_threshold = 0.6
)
print(gap_analysis$summary_stats)
```

---

assess_embedding_stability

Assess Embedding Topic Model Stability

Description

Evaluates the stability of embedding-based topic modeling by running multiple models with different random seeds and comparing their results. High stability (high ARI, consistent keywords) indicates stable topic structure in the data.

Usage

```
assess_embedding_stability(
  texts,
  n_runs = 5,
  embedding_model = "all-MiniLM-L6-v2",
  select_best = TRUE,
  base_seed = 123,
  verbose = TRUE,
  ...
)
```

Arguments

texts	Character vector of documents to analyze.
n_runs	Number of model runs with different seeds (default: 5). More runs give a steadier stability estimate.
embedding_model	Embedding model name (default: "all-MiniLM-L6-v2").
select_best	Logical, if TRUE, returns the best model by quality (default: TRUE).
base_seed	Base random seed; each run uses base_seed + (run - 1).
verbose	Logical, if TRUE, prints progress messages.
...	Additional arguments passed to fit_embedding_model().

Details

Stability is assessed via:

- Adjusted Rand Index (ARI): Measures agreement in topic assignments across runs
- Keyword Jaccard similarity: Measures overlap in top keywords per topic
- Quality variance: Variance in silhouette scores across runs

Value

A list containing:

- stability_metrics: List with mean_ari, sd_ari, mean_jaccard, quality_variance
- best_model: Best model by silhouette score (if select_best = TRUE)
- all_models: List of all fitted models
- is_stable: Logical, TRUE if mean ARI >= 0.6 (considered stable)
- recommendation: Text recommendation based on stability

Examples

```
if (interactive()) {
  texts <- c("Machine learning for image recognition",
             "Deep learning neural networks",
             "Natural language processing models")

  stability <- assess_embedding_stability(
    texts = texts,
    n_runs = 3,
    verbose = TRUE
  )

  # Check if results are stable
  stability$is_stable
  stability$stability_metrics$mean_ari

  # Use the best model
  best_model <- stability$best_model
}
```

`auto_tune_embedding_topics` *Auto-tune BERTopic Hyperparameters*

Description

Automatically searches for optimal hyperparameters for embedding-based topic modeling. Evaluates multiple configurations of UMAP and HDBSCAN parameters and returns the best model based on the specified metric. Embeddings are generated once and reused across all configurations for efficiency.

Usage

```
auto_tune_embedding_topics(
  texts,
  embeddings = NULL,
  embedding_model = "all-MiniLM-L6-v2",
  n_trials = 12,
  metric = "silhouette",
  seed = 123,
  verbose = TRUE
)
```

Arguments

`texts`	Character vector of documents to analyze.
`embeddings`	Precomputed embeddings matrix (optional). If NULL, embeddings are generated.

`embedding_model`
: Embedding model name (default: "all-MiniLM-L6-v2").

`n_trials`
: Maximum number of configurations to try (default: 12).

`metric`
: Optimization metric: "silhouette", "coherence", or "combined" (default: "silhouette").

`seed`
: Random seed for reproducibility.

`verbose`
: Logical, if TRUE, prints progress messages.

Details

The function searches over these parameters:

- n_neighbors: UMAP neighborhood size (5, 10, 15, 25)
- min_cluster_size: HDBSCAN minimum cluster size (3, 5, 10)
- cluster_selection_method: "eom" (broader) or "leaf" (finer-grained)

Value

A list containing:

- best_config: Data frame with the optimal hyperparameter configuration
- best_model: The topic model fitted with optimal parameters
- all_results: List of all evaluated configurations with metrics
- n_trials_completed: Number of configurations successfully evaluated

Examples

```
if (interactive()) {
  texts <- c("Machine learning for image recognition",
             "Deep learning neural networks",
             "Natural language processing models",
             "Computer vision applications")

  tuning_result <- auto_tune_embedding_topics(
    texts = texts,
    n_trials = 6,
    metric = "silhouette",
    verbose = TRUE
  )

  # View best configuration
  tuning_result$best_config

  # Use the best model
  best_model <- tuning_result$best_model
}
```

calculate_assignment_consistency

Calculate Assignment Consistency

Description

Calculates consistency between two sets of assignments

Usage

```
calculate_assignment_consistency(assignments1, assignments2, ...)
```

Arguments

assignments1	First set of assignments
assignments2	Second set of assignments
...	Additional parameters

Value

List containing consistency metrics

calculate_clustering_metrics

Calculate Clustering Quality Metrics

Description

Calculates common clustering evaluation metrics including Silhouette Score, Davies-Bouldin Index, and Calinski-Harabasz Index.

Usage

```
calculate_clustering_metrics(
  clusters,
  data_matrix,
  dist_matrix = NULL,
  metrics = "all"
)
```

Arguments

clusters	Integer vector of cluster assignments
data_matrix	Numeric matrix of data points (rows = observations, cols = features)
dist_matrix	Optional distance matrix. If NULL, computed from data_matrix
metrics	Character vector of metrics to calculate. Options: "silhouette", "davies_bouldin", "calinski_harabasz", or "all" (default)

Details

- Silhouette Score: Measures how similar an object is to its own cluster compared to other clusters. Range: -1 to 1, higher is better.
- Davies-Bouldin Index: Average similarity between each cluster and its most similar cluster. Lower values indicate better clustering.
- Calinski-Harabasz Index: Ratio of between-cluster to within-cluster variance. Higher values indicate better-defined clusters.

Value

A named list containing:

silhouette Silhouette score (-1 to 1, higher is better)

davies_bouldin Davies-Bouldin index (lower is better)

calinski_harabasz Calinski-Harabasz index (higher is better)

n_clusters Number of clusters

cluster_sizes Table of cluster sizes

Examples

```
abstracts <- TextAnalysisR::SpecialEduTech$abstract[1:20]
term_matrix <- as.matrix(quanteda::dfm(quanteda::tokens(abstracts)))
kmeans_result <- stats::kmeans(term_matrix, centers = 2)
metrics <- calculate_clustering_metrics(kmeans_result$cluster, term_matrix)
print(metrics)
```

---

calculate_cross_similarity

Calculate Cross-Matrix Cosine Similarity

Description

Calculates cosine similarity between two different embedding matrices, useful for comparing documents/topics across different categories or groups.

Usage

```
calculate_cross_similarity(
  embeddings1,
  embeddings2,
  labels1 = NULL,
  labels2 = NULL,
  normalize = TRUE
)
```

Arguments

embeddings1	A numeric matrix where rows are items and columns are embedding dimensions.
embeddings2	A numeric matrix where rows are items and columns are embedding dimensions. Must have the same number of columns as embeddings1.
labels1	Optional character vector of labels for items in embeddings1.
labels2	Optional character vector of labels for items in embeddings2.
normalize	Logical, whether to L2-normalize embeddings before computing similarity (default: TRUE).

Value

A list containing:

similarity_matrix	Matrix of cosine similarities (nrow(embeddings1) x nrow(embeddings2))
similarity_df	Long-format data frame with columns: row_idx, col_idx, similarity, and optionally label1, label2

Examples

```
abstracts <- TextAnalysisR::SpecialEduTech$abstract[1:6]
term_matrix <- as.matrix(quanteda::dfm(quanteda::tokens(abstracts)))
similarity_result <- calculate_cross_similarity(
  term_matrix[1:3, ], term_matrix[4:6, ],
  labels1 = paste("Doc", 1:3),
  labels2 = paste("Doc", 4:6)
)
similarity_result$similarity_matrix
```

---

calculate_dispersion_metrics

Calculate Dispersion Metrics

Description

Computes quantitative dispersion metrics for terms, measuring how evenly distributed they are across the corpus.

Usage

```
calculate_dispersion_metrics(tokens_object, terms)
```

Arguments

tokens_object	A quanteda tokens object
terms	Character vector of terms to analyze

Value

Data frame with columns:

- term: The search term
- frequency: Total occurrences
- doc_count: Number of documents containing term
- doc_ratio: Proportion of documents containing term
- juilland_d: Juilland's D dispersion (0-1, higher = more even), computed on within-document proportions
- rosengren_s: Rosengren's S dispersion adjusted for document sizes (1/n_docs to 1, higher = more even)

`calculate_keyword_stability` *Calculate Keyword Stability*

Description

Calculates stability between the topic keyword sets of two independently fitted models. Topic numbering is arbitrary across fits, so topics are matched one-to-one by greedy assignment on the pairwise keyword Jaccard matrix before averaging matched-pair similarity.

Usage

```
calculate_keyword_stability(keywords1, keywords2)
```

Arguments

`keywords1`	List of keyword vectors, one per topic, from the first model.
`keywords2`	List of keyword vectors, one per topic, from the second model.

Value

Mean Jaccard similarity of matched topic pairs (0-1).

`calculate_lexical_dispersion` *Calculate Lexical Dispersion*

Description

Computes lexical dispersion data for specified terms across a corpus. Shows where terms appear within each document, useful for understanding term distribution patterns.

Usage

```
calculate_lexical_dispersion(
  tokens_object,
  terms,
  scale = c("relative", "absolute")
)
```

Arguments

`tokens_object`	A quanteda tokens object
`terms`	Character vector of terms to analyze
`scale`	Character, "relative" (0-1 normalized) or "absolute" (token position)

Value

Data frame with columns:

- doc_id: Document identifier
- term: The search term
- position: Position in document (relative or absolute)
- doc_length: Total tokens in document

Examples

```
tokens <- quanteda::tokens(TextAnalysisR::SpecialEduTech$abstract[1:5])
dispersion <- calculate_lexical_dispersion(tokens, c("learning", "instruction"))
```

`calculate_log_odds_ratio` *Calculate Log Odds Ratio Between Categories*

Description

Compares word usage between categories using the log-odds ratio with a uniform Dirichlet prior, ranked by z-score. Identifies words distinctively used in one category. For the informative-prior weighted log-odds, see `calculate_weighted_log_odds()`.

Usage

```
calculate_log_odds_ratio(
  dfm_object,
  group_var,
  comparison_mode = c("binary", "one_vs_rest", "pairwise"),
  reference_level = NULL,
  top_n = 10,
  min_count = 5
)
```

Arguments

`dfm_object`	A quanteda dfm object
`group_var`	Character, name of the grouping variable in docvars
`comparison_mode`	Character, one of "binary", "one_vs_rest", or "pairwise" • binary: Compare two categories directly • one_vs_rest: Compare each category against all others combined • pairwise: Compare all pairs of categories
`reference_level`	Character, reference category for binary comparison (default: first level)
`top_n`	Number of top terms per comparison (default: 10)
`min_count`	Minimum word count to include (default: 5)

Value

Data frame with columns:

- term: The word/feature
- category1: First category in comparison
- category2: Second category in comparison
- count1: Count in category 1
- count2: Count in category 2

- odds1: Smoothed odds in category 1, (count + 1) / (total + V - count - 1)
- odds2: Smoothed odds in category 2
- odds_ratio: Ratio of odds
- log_odds_ratio: Log of odds ratio (positive = more in compared category)
- variance: Approximate variance of the log odds ratio
- z_score: Log odds ratio divided by its standard error
- significant: TRUE when |z| >= 1.96

Terms are ranked by absolute z-score.

Examples

```
articles <- TextAnalysisR::SpecialEduTech[1:20, ]
corpus <- quanteda::corpus(
  articles$abstract,
  docvars = data.frame(reference_type = articles$reference_type)
)
dfm_object <- quanteda::dfm(quanteda::tokens(corpus))
log_odds <- calculate_log_odds_ratio(dfm_object, "reference_type")
```

`calculate_semantic_drift` *Calculate Semantic Drift*

Description

Calculates semantic drift across time periods

Usage

```
calculate_semantic_drift(temporal_results, ...)
```

Arguments

`temporal_results`	Temporal analysis results
`...`	Additional parameters

Value

List containing drift metrics

calculate_similarity_robust *Calculate Document Similarity with Fallbacks*

Description

Calculates document similarity with fallback methods and diagnostics. Attempts embeddings first, falls back to Jaccard similarity if needed.

Usage

```
calculate_similarity_robust(
  texts,
  method = "embeddings",
  embedding_model = "all-MiniLM-L6-v2",
  cache_embeddings = TRUE,
  min_word_length = 3,
  doc_names = NULL
)
```

Arguments

texts	Character vector of texts
method	Similarity method ("embeddings" or "jaccard")
embedding_model	Model name for embeddings (default: "all-MiniLM-L6-v2")
cache_embeddings	Logical, cache embeddings (default: TRUE)
min_word_length	Minimum word length for Jaccard (default: 3)
doc_names	Optional document names

Value

List containing similarity matrix, method used, embeddings, and diagnostics

Examples

```
if (interactive()) {
  abstracts <- TextAnalysisR::SpecialEduTech$abstract[1:5]
  similarity_result <- calculate_similarity_robust(abstracts)
  print(similarity_result$similarity_matrix)
  print(similarity_result$diagnostics)
}
```

`calculate_text_readability` *Calculate Text Readability*

Description

Calculates multiple readability metrics for texts including Flesch Reading Ease, Flesch-Kincaid Grade Level, Gunning FOG index, and others. Optionally includes lexical diversity metrics and sentence statistics.

Usage

```
calculate_text_readability(
  texts,
  metrics = c("flesch", "flesch_kincaid", "gunning_fog"),
  include_lexical_diversity = TRUE,
  include_sentence_stats = TRUE,
  doc_names = NULL
)
```

Arguments

`texts`	Character vector of texts to analyze
`metrics`	Character vector of readability metrics to calculate. Options: "flesch", "flesch_kincaid", "gunning_fog", "smog", "ari", "coleman_liau"
`include_lexical_diversity`	Logical, include the MTLD lexical diversity index (default: TRUE)
`include_sentence_stats`	Logical, include average sentence length (default: TRUE)
`doc_names`	Optional character vector of document names

Value

A data frame with document names and readability scores

Examples

```
data(SpecialEduTech, package = "TextAnalysisR")
texts <- SpecialEduTech$abstract[1:10]
readability <- calculate_text_readability(texts)
print(readability)
```

calculate_topic_probability *Calculate Topic Probabilities*

Description

Extracts and summarizes topic probabilities (gamma values) from an STM model, returning a formatted data table of mean topic prevalence.

Usage

```
calculate_topic_probability(stm_model, top_n = 10, verbose = TRUE, ...)
```

Arguments

stm_model	A fitted STM model object from stm::stm().
top_n	Number of top topics to display by prevalence (default: 10).
verbose	Logical, if TRUE prints progress messages (default: TRUE).
...	Additional arguments passed to tidytext::tidy().

Value

A DT::datatable showing topics and their mean gamma (prevalence) values, rounded to 3 decimal places.

Examples

```
if (interactive()) {
  data <- TextAnalysisR::SpecialEduTech
  united <- unite_cols(data, c("title", "keyword", "abstract"))
  tokens <- prep_texts(united, text_field = "united_texts")
  dfm_obj <- quanteda::dfm(tokens)
  stm_data <- quanteda::convert(dfm_obj, to = "stm")

  topic_model <- stm::stm(
    documents = stm_data$documents,
    vocab = stm_data$vocab,
    K = 10,
    verbose = FALSE
  )

  prob_table <- calculate_topic_probability(topic_model, top_n = 10)
  print(prob_table)
}
```

`calculate_topic_stability` *Calculate Topic Stability*

Description

Calculates stability of topics across consecutive time periods. Each period is fitted independently, so topics are matched one-to-one on keyword Jaccard similarity before scoring (see `calculate_keyword_stability()`).

Usage

```
calculate_topic_stability(temporal_results)
```

Arguments

`temporal_results`	Results from temporal analysis.

Value

Stability metrics: matched-pair stability per consecutive-period transition and their mean.

`calculate_weighted_log_odds` *Calculate Weighted Log Odds Ratio*

Description

Computes weighted log odds ratios using the method from Monroe, Colaresi, and Quinn (2008) "Fightin' Words" via the tidylo package. This method weights log odds by variance (z-score) to identify words that reliably distinguish between groups, accounting for sampling variability.

Usage

```
calculate_weighted_log_odds(dfm_object, group_var, top_n = 10, min_count = 5)
```

Arguments

`dfm_object`	A quanteda dfm object
`group_var`	Character, name of the document variable to group by
`top_n`	Number of top terms to return per group (default: 10)
`min_count`	Minimum total count for a term to be included (default: 5)

Value

A data frame with the grouping variable, feature, n, log_odds_weighted (from tidylo::bind_log_odds), and significant (TRUE when |log_odds_weighted| >= 1.96)

References

Monroe, B. L., Colaresi, M. P., & Quinn, K. M. (2008). Fightin' words: Lexical feature selection and evaluation for identifying the content of political conflict. Political Analysis, 16(4), 372-403.

Silge, J., & Robinson, D. (2017). Text mining with R: A tidy approach. O'Reilly Media. https://www.tidytextmining.com/

Examples

```
if (requireNamespace("tidylo", quietly = TRUE)) {
  articles <- TextAnalysisR::SpecialEduTech[1:20, ]
  dfm_object <- quanteda::dfm(quanteda::tokens(articles$abstract))
  quanteda::docvars(dfm_object, "reference_type") <- articles$reference_type
  weighted_odds <- calculate_weighted_log_odds(dfm_object, "reference_type",
                                               top_n = 5)
}
```

`calculate_word_frequency`

Analyze and Visualize Word Frequencies Across a Continuous Variable

---

Description

This function analyzes and visualizes word frequencies across a continuous variable.

Usage

```
calculate_word_frequency(
  dfm_object,
  continuous_variable,
  selected_terms,
  height = 500,
  width = 900
)
```

Arguments

`dfm_object`	A quanteda document-feature matrix (dfm).
`continuous_variable`	A continuous variable in the metadata.
`selected_terms`	A vector of terms to analyze trends for.
`height`	The height of the resulting Plotly plot, in pixels (default: 500).
`width`	The width of the resulting Plotly plot, in pixels (default: 900).

Details

This function requires a fitted STM model object and a quanteda dfm object. The continuous variable should be a column in the metadata of the dfm object. The selected terms should be a vector of terms to analyze trends for. The required packages are 'htmltools', 'splines', and 'broom' (plus additional ones loaded internally).

Value

A list containing Plotly objects and tables with the results.

See Also

`extract_keywords_tfidf()` and `extract_keywords_keyness()` for ranking terms by importance; `plot_word_frequency()` for the standard frequency-by-doc plot

Examples

```
mydata <- TextAnalysisR::SpecialEduTech

united_tbl <- TextAnalysisR::unite_cols(
  mydata,
  listed_vars = c("title", "keyword", "abstract")
)

tokens <- TextAnalysisR::prep_texts(united_tbl, text_field = "united_texts")

dfm_object <- quanteda::dfm(tokens)

word_freq_results <- TextAnalysisR::calculate_word_frequency(
  dfm_object,
  continuous_variable = "year",
  selected_terms = c("calculator", "computer"),
  height = 500,
  width = 900
)
print(word_freq_results$plot)
print(word_freq_results$table)
```

---

call_llm_api *Call LLM API (Unified Wrapper)*

Description

Unified wrapper for calling different LLM providers (OpenAI, Gemini). Automatically routes to the appropriate provider-specific function.

Usage

```
call_llm_api(
  provider = c("openai", "gemini"),
  system_prompt,
  user_prompt,
  model = NULL,
  temperature = 0,
  max_tokens = 150,
  api_key = NULL
)
```

Arguments

provider	Character string: "openai" or "gemini"
system_prompt	Character string with system instructions
user_prompt	Character string with user message
model	Character string specifying the model (provider-specific defaults apply)
temperature	Numeric temperature for response randomness (default: 0)
max_tokens	Maximum number of tokens to generate (default: 150)
api_key	Character string with API key (required for openai/gemini)

Value

Character string with the model's response

See Also

sanitize_llm_input() to clean text before prompting; get_best_embeddings() for vector embeddings; run_rag_search() for RAG search (retrieval + generation)

Examples

```
if (interactive()) {
# Using OpenAI
response <- call_llm_api(
  provider = "openai",
  system_prompt = "You are a helpful assistant.",
  user_prompt = "Generate a topic label",
  api_key = Sys.getenv("OPENAI_API_KEY")
)

# Using Gemini
response <- call_llm_api(
  provider = "gemini",
  system_prompt = "You are a helpful assistant.",
  user_prompt = "Generate a topic label",
  api_key = Sys.getenv("GEMINI_API_KEY")
)
}
```

clear_lexdiv_cache *Clear Lexical Diversity Cache*

Description

Clears the internal cache used for lexical diversity calculations. Call this function to free memory or force fresh calculations.

Usage

```
clear_lexdiv_cache()
```

Value

Invisible NULL

cluster_embeddings *Embedding-based Document Clustering*

Description

This function performs clustering analysis using various methods, ordered from simple to detailed: k-means (simplest), hierarchical (intermediate), and UMAP+DBSCAN (most detailed).

Usage

```
cluster_embeddings(
  data_matrix,
  method = "kmeans",
  n_clusters = 0,
  umap_neighbors = 15,
  umap_min_dist = 0.1,
  umap_n_components = 10,
  umap_metric = "cosine",
  dbscan_eps = 0,
  dbscan_min_samples = NULL,
  reduce_outliers = TRUE,
  outlier_strategy = "centroid",
  outlier_threshold = 0,
  seed = 123,
  verbose = TRUE
)
```

Arguments

data_matrix	A numeric matrix where rows represent documents and columns represent features.
method	The clustering method. Options: "kmeans", "hierarchical", "umap_dbscan".
n_clusters	The number of clusters (for k-means and hierarchical). If 0, optimal number is determined automatically.
umap_neighbors	The number of neighbors for UMAP (default: 15).
umap_min_dist	The minimum distance for UMAP (default: 0.1).
umap_n_components	The number of UMAP components (default: 10).
umap_metric	The metric for UMAP (default: "cosine").
dbscan_eps	The eps parameter for DBSCAN. If 0, optimal value is determined automatically.
dbscan_min_samples	The minimum samples for DBSCAN. Default NULL sets umap_n_components + 1 (dimensions-plus-one rule of thumb).
reduce_outliers	Logical, if TRUE, reassigns noise points (cluster 0) to nearest cluster (default: TRUE). Set FALSE to keep density-based noise labels.
outlier_strategy	Strategy for outlier reduction: "centroid" (default, Euclidean nearest-centroid in UMAP space; package-specific heuristic) or "embeddings" (cosine similarity to cluster mean in original embedding space, matching the BERTopic "embeddings" strategy).
outlier_threshold	Minimum similarity for reassigning a noise point (default: 0, reassign all). For "embeddings" this is cosine similarity; for "centroid" it applies to 1 / (1 + euclidean distance). Points below the threshold keep cluster label 0.
seed	Random seed for reproducibility (default: 123).
verbose	Logical, if TRUE, prints progress messages.

Value

A list containing cluster assignments, method used, and quality metrics. For method = "umap_dbscan" with reduce_outliers = TRUE, the list also reports how noise points were reassigned: outlier_strategy ("embeddings" or "centroid"), outlier_metric ("cosine" or "euclidean"), outlier_space ("original embedding space" or "UMAP space"), and outliers_reassigned (count).

Examples

```
if (interactive()) {
  mydata <- TextAnalysisR::SpecialEduTech

  united_tbl <- TextAnalysisR::unite_cols(
    mydata,
```

```
  listed_vars = c("title", "keyword", "abstract")
)

tokens <- TextAnalysisR::prep_texts(united_tbl, text_field = "united_texts")

dfm_object <- quanteda::dfm(tokens)

data_matrix <- as.matrix(dfm_object)

kmeans_result <- TextAnalysisR::cluster_embeddings(
  data_matrix,
  method = "kmeans",
  n_clusters = 5
)
print(kmeans_result)

hierarchical_result <- TextAnalysisR::cluster_embeddings(
  data_matrix,
  method = "hierarchical",
  n_clusters = 5
)
print(hierarchical_result)

umap_dbscan_result <- TextAnalysisR::cluster_embeddings(
  data_matrix,
  method = "umap_dbscan",
  umap_neighbors = 15,
  umap_min_dist = 0.1
)
print(umap_dbscan_result)
}
```

`describe_image` *Describe Image Using Vision LLM*

Description

Unified dispatcher for image description using vision LLMs. Routes to the appropriate provider (OpenAI or Gemini).

Usage

```
describe_image(
  image_base64,
  provider = "gemini",
  model = NULL,
  api_key = NULL,
  prompt =
   "Describe this image: charts, diagrams, tables, and text. Extract visible text.",
  timeout = 120
)
```

Arguments

image_base64	Character string of base64-encoded PNG image
provider	Character: "openai" or "gemini"
model	Character: Model name (uses provider default if NULL)
api_key	Character: API key (required for openai/gemini)
prompt	Character: Description prompt
timeout	Numeric: Request timeout in seconds (default: 120)

Value

Character string description, or NULL on failure

---

detect_multi_words *Detect Multi-Word Expressions*

Description

This function detects multi-word expressions (collocations) of specified sizes that appear at least a specified number of times in the provided tokens.

Usage

```
detect_multi_words(tokens, size = 2:5, min_count = 2)
```

Arguments

tokens	A tokens object from the quanteda package.
size	A numeric vector specifying the sizes of the collocations to detect (default: 2:5).
min_count	The minimum number of occurrences for a collocation to be considered (default: 2).

Value

A character vector of detected collocations.

Examples

```
mydata <- TextAnalysisR::SpecialEduTech[seq_len(50), ]

united_tbl <- TextAnalysisR::unite_cols(
  mydata,
  listed_vars = c("title", "keyword", "abstract")
)

tokens <- TextAnalysisR::prep_texts(united_tbl, text_field = "united_texts")
```

```
collocations <- TextAnalysisR::detect_multi_words(tokens, size = 2:3, min_count = 2)
print(collocations)
```

`detect_pdf_content_type` *Detect PDF Content Type*

Description

Analyzes PDF to determine if it contains readable text.

Usage

```
detect_pdf_content_type(file_path)
```

Arguments

`file_path`	Character string path to PDF file

Details

Attempts text extraction using pdftools. Returns "text" if successful, or "unknown" if extraction fails or PDF is empty.

For table extraction from PDFs, use `extract_tables_from_pdf_py`.

Value

Character string: "text" or "unknown"

Examples

```
pdf_path <- "path/to/document.pdf"
content_type <- detect_pdf_content_type(pdf_path)
print(content_type)
```

export_document_clustering

Export Document Clustering Analysis

Description

Export document clustering analysis results to CSV

Usage

```
export_document_clustering(
  coordinates,
  clusters = NULL,
  labels = NULL,
  doc_ids = NULL,
  file_path
)
```

Arguments

coordinates	Document coordinates
clusters	Cluster assignments (optional)
labels	Cluster labels (optional)
doc_ids	Document IDs
file_path	Path to save the CSV file

Value

Invisible data frame of the exported data

extract_cross_category_similarities

Extract Cross-Category Similarities from Full Similarity Matrix

Description

Given a full similarity matrix and category information, extracts pairwise similarities between a reference category and other categories into a long-format data frame suitable for visualization and analysis.

Usage

```
extract_cross_category_similarities(
  similarity_matrix,
  docs_data,
  reference_category,
  compare_categories = NULL,
  category_var = "category",
  id_var = "display_name",
  name_var = NULL
)
```

Arguments

similarity_matrix	A square similarity matrix (n x n).
docs_data	A data frame containing document metadata with at least: **category_var** Column indicating category membership **id_var** Column with unique document identifiers
reference_category	Character string specifying the reference category to compare against.
compare_categories	Character vector of categories to compare with the reference. If NULL, compares with all categories except reference.
category_var	Name of the column containing category information (default: "category").
id_var	Name of the column containing document IDs (default: "display_name").
name_var	Optional name of column with display names (default: NULL, uses id_var).

Value

A data frame with columns:

ref_id	Reference document ID
ref_name	Reference document name (if name_var provided)
other_id	Comparison document ID
other_name	Comparison document name (if name_var provided)
other_category	Category of comparison document
similarity	Cosine similarity value

Examples

```
articles <- TextAnalysisR::SpecialEduTech[1:6, ]
articles$display_name <- paste0("d", seq_len(nrow(articles)))
term_matrix <- as.matrix(quanteda::dfm(quanteda::tokens(articles$abstract)))
normalized_matrix <- term_matrix / sqrt(rowSums(term_matrix ^ 2))
similarity_matrix <- normalized_matrix %*% t(normalized_matrix)
dimnames(similarity_matrix) <- list(articles$display_name, articles$display_name)
```

```
cross_similarities <- extract_cross_category_similarities(
  similarity_matrix  = similarity_matrix,
  docs_data          = articles,
  reference_category = "thesis",
  compare_categories = "journal_article",
  category_var       = "reference_type",
  id_var             = "display_name",
  name_var           = "title"
)
```

`extract_keywords_keyness` *Extract Keywords Using Statistical Keyness*

Description

Extracts distinctive keywords by comparing document groups using log-likelihood ratio (G-squared).

Usage

```
extract_keywords_keyness(
  dfm,
  target,
  top_n = 20,
  measure = "lr",
  min_count = 0
)
```

Arguments

`dfm`	A quanteda dfm object
`target`	Target document indices or logical vector
`top_n`	Number of top keywords to extract (default: 20)
`measure`	Keyness measure: "lr" (log-likelihood G-squared), "chi2", "exact" (Fisher's exact odds ratio), or "pmi" (default: "lr")
`min_count`	Minimum total term frequency before computing keyness (default: 0, no pruning)

Value

Data frame with columns: Keyword, Keyness_Score

Examples

```
abstracts <- TextAnalysisR::SpecialEduTech$abstract[1:10]
dfm_object <- quanteda::dfm(quanteda::tokens(quanteda::corpus(abstracts)))
keywords <- extract_keywords_keyness(dfm_object, target = 1)
print(keywords)
```

`extract_keywords_tfidf`

Extract Keywords Using TF-IDF

Description

Extracts top keywords from a document-feature matrix using TF-IDF weighting. Uses quanteda's default scheme (raw term counts, base-10 idf), which differs from tidytext's proportion-based tf; rankings favor terms in longer documents.

Usage

```
extract_keywords_tfidf(dfm, top_n = 20, normalize = FALSE)
```

Arguments

`dfm`	A quanteda dfm object
`top_n`	Number of top keywords to extract (default: 20)
`normalize`	Logical, whether to normalize TF-IDF scores to 0-1 range (default: FALSE)

Value

Data frame with columns: Keyword, TF_IDF_Score, Frequency

Examples

```
library(quanteda)
corp <- corpus(c("text analysis", "data mining", "text mining"))
dfm_obj <- dfm(tokens(corp))
keywords <- extract_keywords_tfidf(dfm_obj, top_n = 5)
print(keywords)
```

`extract_morphology` *Extract Morphological Features*

Description

Uses spaCy to extract morphological features from text. Returns data with Number, Tense, VerbForm, Person, Case, Mood, Aspect, etc.

Usage

```
extract_morphology(
  tokens,
  features = c("Number", "Tense", "VerbForm", "Person", "Case", "Mood", "Aspect"),
  include_pos = TRUE,
  include_lemma = TRUE,
  model = "en_core_web_sm"
)
```

Arguments

`tokens`	A quanteda tokens object or character vector of texts.
`features`	Character vector of morphological features to extract. Default includes common Universal Dependencies features.
`include_pos`	Logical; include POS tags (default: TRUE).
`include_lemma`	Logical; include lemmatized forms (default: TRUE).
`model`	Character; spaCy model to use (default: "en_core_web_sm").

Details

Morphological features follow Universal Dependencies annotation. Common features include:

- `Number`: Sing (singular), Plur (plural)
- `Tense`: Past, Pres (present), Fut (future)
- `VerbForm`: Fin (finite), Inf (infinitive), Part (participle), Ger (gerund)
- `Person`: 1, 2, 3 (first, second, third person)
- `Case`: Nom (nominative), Acc (accusative), Gen (genitive), Dat (dative)
- `Mood`: Ind (indicative), Imp (imperative), Sub (subjunctive)
- `Aspect`: Perf (perfective), Imp (imperfective), Prog (progressive)

Value

A data frame with token-level morphological annotations including morph_* columns for each requested feature.

Examples

```
if (interactive()) {
  tokens <- quanteda::tokens(TextAnalysisR::SpecialEduTech$abstract[1])
  morphology_data <- extract_morphology(tokens)
  print(morphology_data)
}
```

`extract_named_entities` *Extract Named Entities from Tokens*

Description

Uses spaCy to extract named entities (NER) from tokenized text. Returns a data frame with token-level entity annotations.

Usage

```
extract_named_entities(
  tokens,
  include_pos = TRUE,
  include_lemma = TRUE,
  model = "en_core_web_sm"
)
```

Arguments

`tokens`	A quanteda tokens object or character vector of texts.
`include_pos`	Logical; include POS tags (default: TRUE).
`include_lemma`	Logical; include lemmatized forms (default: TRUE).
`model`	Character; spaCy model to use (default: "en_core_web_sm").

Details

This function requires the Python with spaCy installed. If spaCy is not initialized, this function will attempt to initialize it with the specified model.

Value

A data frame with columns:

- `doc_id`: Document identifier
- `token`: Original token
- `entity`: Named entity type (e.g., PERSON, ORG, GPE)
- `pos`: Universal POS tag (if include_pos = TRUE)
- `lemma`: Lemmatized form (if include_lemma = TRUE)

Examples

```
if (interactive()) {
  tokens <- quanteda::tokens(TextAnalysisR::SpecialEduTech$abstract[1])
  entity_data <- extract_named_entities(tokens)
  print(entity_data)
}
```

---

extract_noun_chunks *Extract Noun Chunks*

Description

Extract noun chunks (base noun phrases) from texts. Useful for keyphrase extraction.

Usage

```
extract_noun_chunks(x, model = "en_core_web_sm")
```

Arguments

x	Character vector of texts OR a quanteda tokens object.
model	Character; spaCy model to use (default: "en_core_web_sm").

Value

A data frame with noun chunk information.

---

extract_pos_tags *Extract Part-of-Speech Tags from Tokens*

Description

Uses spaCy to extract part-of-speech (POS) tags from tokenized text. Returns a data frame with token-level POS annotations.

Usage

```
extract_pos_tags(
  tokens,
  include_lemma = TRUE,
  include_entity = FALSE,
  include_dependency = FALSE,
  model = "en_core_web_sm"
)
```

Arguments

tokens	A quanteda tokens object or character vector of texts.
include_lemma	Logical; include lemmatized forms (default: TRUE).
include_entity	Logical; include named entity recognition (default: FALSE).
include_dependency	Logical; include dependency parsing (default: FALSE).
model	Character; spaCy model to use (default: "en_core_web_sm").

Details

This function requires the Python with spaCy installed. If spaCy is not initialized, this function will attempt to initialize it with the specified model.

Value

A data frame with columns:

- doc_id: Document identifier
- sentence_id: Sentence number within document
- token_id: Token position within sentence
- token: Original token
- pos: Universal POS tag (e.g., NOUN, VERB, ADJ)
- tag: Detailed POS tag (e.g., NN, VBD, JJ)
- lemma: Lemmatized form (if include_lemma = TRUE)
- entity: Named entity type (if include_entity = TRUE)
- head_token_id: Head token in dependency tree (if include_dependency = TRUE)
- dep_rel: Dependency relation type, e.g., nsubj, dobj (if include_dependency = TRUE)

Examples

```
if (interactive()) {
  tokens <- quanteda::tokens(TextAnalysisR::SpecialEduTech$abstract[1])
  pos_data <- extract_pos_tags(tokens)
  print(pos_data)
}
```

extract_subjects_objects *Extract Subjects and Objects*

Description

Extract subject-verb-object (SVO) triples from texts using dependency parsing.

Usage

```
extract_subjects_objects(x, model = "en_core_web_sm")
```

Arguments

x	Character vector of texts OR a quanteda tokens object.
model	Character; spaCy model to use (default: "en_core_web_sm").

Value

A data frame with SVO information.

extract_topic_terms_df *Build a topic-term data frame from any supported topic model*

Description

Unified helper that produces the long-format `data.frame(topic, term, beta)` expected by `generate_topic_labels()` from an STM model or an embedding result. Dispatches on the object's structure:

- STM model (has `$beta$logbeta` and `$vocab`) -> top terms via `stm::labelTopics()` FREX
- Embedding result (has `$topic_keywords`) -> c-TF-IDF keywords with rank-derived pseudo-beta

Usage

```
extract_topic_terms_df(model, n = 7)
```

Arguments

model	A topic model object (STM fit or embedding result).
n	Number of top terms per topic (default 7).

Value

`data.frame(topic, term, beta)` in long format.

`find_optimal_k` *Find Optimal Number of Topics*

Description

Searches for the optimal number of topics (K) using stm::searchK. Produces diagnostic plots to help select the best K value. The default spectral initialization is deterministic; with LDA or random initialization, fit several seeds per K before selecting a model.

Usage

```
find_optimal_k(
  dfm_object,
  topic_range,
  max.em.its = 75,
  emtol = 1e-04,
  cores = 1,
  categorical_var = NULL,
  continuous_var = NULL,
  height = 600,
  width = 800,
  verbose = TRUE,
  ...
)
```

Arguments

`dfm_object`	A quanteda dfm object to be used for topic modeling.
`topic_range`	A vector of K values to test (e.g., 2:10).
`max.em.its`	Maximum number of EM iterations (default: 75).
`emtol`	Convergence tolerance for EM algorithm (default: 1e-04). Higher values (e.g., 1e-03) speed up fitting but may reduce precision.
`cores`	Number of CPU cores to use for parallel processing (default: 1). Set to higher values for faster searchK on multi-core systems.
`categorical_var`	Optional categorical variable(s) for prevalence.
`continuous_var`	Optional continuous variable(s) for prevalence.
`height`	Plot height in pixels (default: 600).
`width`	Plot width in pixels (default: 800).
`verbose`	Logical indicating whether to print progress (default: TRUE).
`...`	Additional arguments passed to stm::searchK.

Value

A list containing search results and diagnostic plots.

See Also

plot_quality_metrics() to visualize topic-count diagnostics; stm::stm() to fit the chosen model; fit_embedding_model() for an embedding-based alternative to STM

---

find_similar_words *Find Similar Words*

Description

Find words most similar to a given word using word vectors. Requires a spaCy model with word vectors (en_core_web_md or en_core_web_lg).

Usage

```
find_similar_words(word, top_n = 10L, model = "en_core_web_md")
```

Arguments

word	Character; target word.
top_n	Integer; number of similar words to return (default: 10).
model	Character; spaCy model to use (default: "en_core_web_md").

Value

A data frame with similar words and similarity scores.

---

find_topic_matches *Find Similar Topics*

Description

This function finds the most similar topics to a given query using semantic similarity analysis. It works with both semantic topic models and traditional STM models by creating topic representations using transformer embeddings and calculating cosine similarity scores.

Usage

```
find_topic_matches(
  topic_model,
  query,
  top_n = 10,
  method = "cosine",
  embedding_model = "all-MiniLM-L6-v2",
  include_terms = TRUE
)
```

Arguments

topic_model	A topic model object (semantic topic model or STM model).
query	A character string representing the query topic.
top_n	The number of similar topics to return (default: 10).
method	The similarity method: "cosine", "euclidean", "embedding".
embedding_model	The embedding model to use for query encoding (default: "all-MiniLM-L6-v2").
include_terms	Logical, whether to include topic terms in the similarity calculation (default: TRUE).

Value

A list containing similar topics and their similarity scores.

Examples

```
if (interactive()) {
  mydata <- TextAnalysisR::SpecialEduTech
  united_tbl <- TextAnalysisR::unite_cols(
    mydata,
    listed_vars = c("title", "keyword", "abstract")
  )
  texts <- united_tbl$united_texts

  topic_model <- TextAnalysisR::fit_embedding_model(
    texts = texts,
    method = "umap_hdbscan",
    n_topics = 8
  )

  similar_topics <- TextAnalysisR::find_topic_matches(
    topic_model = topic_model,
    query = "mathematical learning",
    top_n = 5
  )

  print(similar_topics)
}
```

---

fit_embedding_model *Fit Embedding-based Topic Model*

Description

This function performs embedding-based topic modeling using transformer embeddings and specialized clustering techniques. Supports two backends:

- **Python backend** (default): Uses BERTopic library which combines transformer embeddings with UMAP dimensionality reduction and HDBSCAN clustering for optimal topic discovery.
- **R backend**: Uses R-native packages (umap, dbscan, Rtsne) for users without Python/BERTopic installed. Provides similar functionality with c-TF-IDF keyword extraction.

Usage

```
fit_embedding_model(
  texts,
  method = "umap_hdbscan",
  n_topics = 10,
  embedding_model = "all-MiniLM-L6-v2",
  backend = "auto",
  clustering_method = "kmeans",
  similarity_threshold = 0.7,
  min_topic_size = 10,
  min_cluster_size = NULL,
  cluster_selection_method = "eom",
  umap_neighbors = 15,
  umap_min_dist = 0,
  umap_n_components = 5,
  umap_metric = "cosine",
  tsne_perplexity = 30,
  pca_dims = 50,
  dbscan_eps = 0.5,
  dbscan_minpts = 5,
  representation_method = "c-tfidf",
  diversity = 0.5,
  reduce_outliers = TRUE,
  outlier_strategy = "probabilities",
  outlier_threshold = 0,
  seed = 123,
  verbose = TRUE,
  precomputed_embeddings = NULL
)
```

Arguments

texts	A character vector of texts to analyze.
method	The topic modeling method: • For Python backend: "umap_hdbscan" (uses BERTopic) • For R backend: "umap_dbscan", "umap_kmeans", "umap_hierarchical", "tsne_dbscan", "tsne_kmeans", "pca_kmeans", "pca_hierarchical" • For both: "embedding_clustering", "hierarchical_semantic"

`n_topics` The number of topics to identify. For UMAP+HDBSCAN, use NULL or "auto" for automatic determination, or specify an integer.

`embedding_model`
The embedding model to use (default: "all-MiniLM-L6-v2").

`backend` The backend to use: "auto" (default, tries Python then R), "python" (requires BERTopic), or "r" (R-native packages only).

`clustering_method`
The clustering method for embedding-based approach: "kmeans", "hierarchical", "dbscan", "hdbscan".

`similarity_threshold`
The similarity threshold for topic assignment (default: 0.7).

`min_topic_size` The minimum number of documents per topic (default: 3).

`min_cluster_size`
HDBSCAN density threshold (default `NULL` falls back to `min_topic_size`). Setting this independently lets fine-grained clusters merge into broader topics.

`cluster_selection_method`
HDBSCAN cluster selection method: "eom" (Excess of Mass, default) or "leaf" (finer-grained topics).

`umap_neighbors` The number of neighbors for UMAP dimensionality reduction (default: 15).

`umap_min_dist` The minimum distance for UMAP (default: 0.0). Use 0.0 for tight, well-separated clusters. Use 0.1+ for visualization purposes. Range: 0.0-0.99.

`umap_n_components`
The number of UMAP components (default: 5).

`umap_metric` Distance metric for UMAP: "cosine" (recommended for text) or "euclidean" (default: "cosine").

`tsne_perplexity`
Perplexity parameter for t-SNE (default: 30). Only used when method includes "tsne"; t-SNE output is 2-dimensional.

`pca_dims` Number of PCA components kept for clustering (default: 50). Only used when method includes "pca".

`dbscan_eps` Epsilon parameter for DBSCAN (default: 0.5). Neighborhood size for density-based clustering.

`dbscan_minpts` Minimum points for DBSCAN core points (default: 5).

`representation_method`
The method for topic representation: "c-tfidf", "tfidf", "mmr", "frequency" (default: "c-tfidf"). Applies only to the R fallback backend and the "embedding_clustering", "semantic_lda", and "hierarchical_semantic" methods; the Python BERTopic backend ("umap_hdbscan") uses BERTopic's native c-TF-IDF representation.

`diversity` Diversity weight between 0 and 1 for the "mmr" representation (default: 0.5). Higher values penalize redundant terms more strongly. Applies to the R backend; ignored by the Python BERTopic backend.

`reduce_outliers`
Logical, if TRUE, reduces outliers in HDBSCAN clustering (default: TRUE).

`outlier_strategy`
: Strategy for outlier reduction using BERTopic: "probabilities" (default, uses topic probabilities), "c-tf-idf" (uses c-TF-IDF similarity), "embeddings" (uses cosine similarity in embedding space), or "distributions" (uses topic distributions). Ignored if reduce_outliers = FALSE.

`outlier_threshold`
: Minimum threshold for outlier reassignment (default: 0.0). Higher values require stronger evidence for reassignment.

`seed`
: Random seed for reproducibility (default: 123).

`verbose`
: Logical, if TRUE, prints progress messages.

`precomputed_embeddings`
: Optional matrix of pre-computed document embeddings. If provided, skips embedding generation for improved performance. Must have the same number of rows as the length of texts.

Value

A list containing topic assignments, topic keywords, and quality metrics.

See Also

`get_best_embeddings()` to supply precomputed embeddings; `generate_topic_labels()` for AI-suggested topic names; `find_optimal_k()` for an STM-based alternative

Examples

```
if (interactive()) {
  mydata <- TextAnalysisR::SpecialEduTech
  united_tbl <- TextAnalysisR::unite_cols(
    mydata,
    listed_vars = c("title", "keyword", "abstract")
  )
  texts <- united_tbl$united_texts

  # Embedding-based topic modeling (powered by BERTopic)
  result <- TextAnalysisR::fit_embedding_model(
    texts = texts,
    method = "umap_hdbscan",
    n_topics = 8,
    min_topic_size = 3
  )

  print(result$topic_assignments)
  print(result$topic_keywords)
}
```

`fit_semantic_model` *Fit Semantic Model*

Description

Performs semantic analysis including similarity, dimensionality reduction, and clustering. This is a high-level wrapper function.

Usage

```
fit_semantic_model(
  texts,
  analysis_types = c("similarity", "dimensionality_reduction", "clustering"),
  document_feature_type = "embeddings",
  similarity_method = "cosine",
  use_embeddings = TRUE,
  embedding_model = "all-MiniLM-L6-v2",
  dimred_method = "UMAP",
  clustering_method = "umap_dbscan",
  n_components = 2,
  n_clusters = 5,
  seed = 123,
  verbose = TRUE
)
```

Arguments

`texts`	A character vector of texts to analyze.
`analysis_types`	Types of analysis to perform: "similarity", "dimensionality_reduction", "clustering".
`document_feature_type`	Feature extraction type (default: "embeddings").
`similarity_method`	Similarity calculation method (default: "cosine").
`use_embeddings`	Logical, use embedding-based approaches (default: TRUE).
`embedding_model`	Sentence transformer model name (default: "all-MiniLM-L6-v2").
`dimred_method`	Dimensionality reduction method: "PCA", "t-SNE", "UMAP" (default: "UMAP").
`clustering_method`	Clustering method: "kmeans", "hierarchical", "umap_dbscan" (default: "umap_dbscan").
`n_components`	Number of dimensions for reduction (default: 2).
`n_clusters`	Number of clusters (default: 5).
`seed`	Random seed for reproducibility (default: 123).
`verbose`	Logical, if TRUE, prints progress messages.

Value

A list containing results from requested analyses.

Examples

```
if (interactive()) {
  data(SpecialEduTech)
  texts <- SpecialEduTech$abstract[1:10]

  results <- fit_semantic_model(
    texts = texts,
    analysis_types = c("similarity", "clustering")
  )

  print(results)
}
```

---

fit_temporal_model *Fit Temporal Topic Model*

Description

Analyzes how topics evolve over time by fitting topic models to different time periods and tracking semantic changes.

Usage

```
fit_temporal_model(
  texts,
  dates,
  time_windows = "yearly",
  embeddings = NULL,
  verbose = TRUE
)
```

Arguments

texts	A character vector of text documents to analyze.
dates	A vector of dates corresponding to each document (will be converted to Date).
time_windows	Time grouping strategy: "yearly", "monthly", or "quarterly" (default: "yearly").
embeddings	Optional pre-computed embeddings matrix. If NULL, embeddings will be generated.
verbose	Logical indicating whether to print progress messages (default: TRUE).

Value

A list containing temporal analysis results with topic evolution patterns.

generate_cluster_labels *Generate Cluster Label Suggestions (Human-in-the-Loop)*

Description

Suggests descriptive labels for clusters using AI. Labels are suggestions for human review - users should edit and approve before using. Supports OpenAI or Gemini for AI generation.

Usage

```
generate_cluster_labels(
  cluster_keywords,
  provider = "auto",
  model = NULL,
  temperature = 0.3,
  max_tokens = 50,
  api_key = NULL,
  verbose = TRUE
)
```

Arguments

cluster_keywords	List of keywords for each cluster.
provider	AI provider to use: "auto" (default), "openai", or "gemini". "auto" picks the first provider with an available API key.
model	Model name. If NULL, uses provider defaults: "gpt-4.1-mini" (OpenAI), "gemini-2.5-flash-lite" (Gemini).
temperature	Temperature parameter (default: 0.3).
max_tokens	Maximum tokens for response (default: 50).
api_key	API key for OpenAI or Gemini. If NULL, uses environment variable.
verbose	Logical, if TRUE, prints progress messages.

Value

A list of generated labels.

Examples

```
if (interactive()) {
  cluster_keywords <- list(
    "1" = c("calculator", "arithmetic", "elementary", "remedial"),
    "2" = c("computer", "instruction", "multiplication", "drill")
  )
  labels_openai <- generate_cluster_labels(cluster_keywords, provider = "openai")
```

```
  labels_gemini <- generate_cluster_labels(cluster_keywords, provider = "gemini")
}
```

`generate_cluster_labels_auto` *Generate Cluster Labels*

Description

Generate descriptive labels for document clusters

Usage

```
generate_cluster_labels_auto(
  feature_matrix,
  clusters,
  method = "tfidf",
  n_terms = 3
)
```

Arguments

`feature_matrix`	Feature matrix used for clustering
`clusters`	Cluster assignments
`method`	Label generation method ("tfidf", "representative", "frequent")
`n_terms`	Number of terms per label

Value

Named list of cluster labels

`generate_embeddings` *Generate Embeddings*

Description

Generates L2-normalized embeddings for texts using sentence transformers.

Usage

```
generate_embeddings(texts, model = "all-MiniLM-L6-v2", verbose = TRUE)
```

Arguments

texts	A character vector of texts.
model	Sentence transformer model name (default: "all-MiniLM-L6-v2"). Newer small models (e.g. "BAAI/bge-small-en-v1.5") often score higher on retrieval benchmarks but expect instruction prefixes this function does not add.
verbose	Logical, if TRUE, prints progress messages.

Value

A matrix of L2-normalized embeddings.

---

generate_topic_content

Generate Content from Topic Terms

Description

Uses Large Language Models (LLMs) to generate various types of content based on topic model terms. Supports multiple content types with optimized default prompts, or fully custom prompts.

Usage

```
generate_topic_content(
  topic_terms_df,
  content_type = c("survey_item", "research_question", "theme_description",
    "policy_recommendation", "interview_question", "custom"),
  topic_var = "topic",
  term_var = "term",
  weight_var = "beta",
  provider = c("openai", "gemini"),
  model = "gpt-4.1-mini",
  temperature = 0,
  system_prompt = NULL,
  user_prompt_template = NULL,
  max_tokens = 150,
  api_key = NULL,
  output_var = NULL,
  verbose = TRUE
)
```

Arguments

topic_terms_df	A data frame with topic terms, containing columns for topic identifier, term, and optionally term weight (beta).
content_type	Type of content to generate. One of:
	"survey_item" Likert-scale survey items for scale development

"research_question" Research questions for literature review
"theme_description" Theme descriptions for qualitative analysis
"policy_recommendation" Policy recommendations for policy analysis
"interview_question" Interview questions for qualitative research
"custom" Custom content using user-provided prompts

`topic_var` Name of the column containing topic identifiers (default: "topic").

`term_var` Name of the column containing terms (default: "term").

`weight_var` Name of the column containing term weights (default: "beta").

`provider` LLM provider: "openai" or "gemini" (default: "openai").

`model` Model name. For OpenAI: "gpt-4.1-mini", "gpt-4", etc. For Gemini: "gemini-2.5-flash-lite", "gemini-2.5-flash", etc.

`temperature` Sampling temperature (0-2). Lower = more deterministic (default: 0).

`system_prompt` Custom system prompt. If NULL, uses default for content_type.

`user_prompt_template`
Custom user prompt template with {terms} placeholder. If NULL, uses default for content_type.

`max_tokens` Maximum tokens for response (default: 150).

`api_key` API key for the selected provider. If NULL, reads from OPENAI_API_KEY or GEMINI_API_KEY environment variable.

`output_var` Name of the output column (default: based on content_type).

`verbose` Logical, if TRUE, prints progress messages.

Details

The function generates one piece of content per unique topic. Each content type has optimized default prompts, but these can be overridden with custom prompts.

Requires an API key set via the `api_key` parameter or the relevant environment variable (OPENAI_API_KEY or GEMINI_API_KEY).

Value

A data frame with generated content joined to original topic terms.

See Also

`generate_topic_labels()` for the step that creates topic labels; `get_content_type_prompt()` and `get_content_type_user_template()` to inspect or override default prompts

Examples

```
if (interactive()) {
# Generate survey items
survey_items <- generate_topic_content(
  topic_terms_df = top_terms,
  content_type = "survey_item",
```

```
  provider = "openai",
  model = "gpt-4.1-mini"
)

# Generate research questions
research_qs <- generate_topic_content(
  topic_terms_df = top_terms,
  content_type = "research_question",
  provider = "gemini",
  model = "gemini-2.5-flash-lite"
)

# Generate with custom prompt
custom_content <- generate_topic_content(
  topic_terms_df = top_terms,
  content_type = "custom",
  system_prompt = "You are an expert in educational policy...",
  user_prompt_template = "Based on {terms}, generate a learning objective:"
)
}
```

---

generate_topic_labels *Generate Topic Labels Using AI*

Description

This function generates descriptive labels for each topic based on their top terms using AI providers (OpenAI or Gemini).

Usage

```
generate_topic_labels(
  top_topic_terms,
  provider = "auto",
  model = NULL,
  system = NULL,
  user = NULL,
  temperature = 0.5,
  api_key = NULL,
  openai_api_key = NULL,
  verbose = TRUE
)
```

Arguments

top_topic_terms	A data frame containing the top terms for each topic.
provider	AI provider to use: "auto" (default), "openai", or "gemini". "auto" picks the first provider with an available API key.

model	A character string specifying which model to use. If NULL, uses provider defaults: "gpt-4.1-mini" (OpenAI), "gemini-2.5-flash-lite" (Gemini).
system	A character string containing the system prompt for the API. If NULL, the function uses the default system prompt.
user	A character string containing the user prompt for the API. If NULL, the function uses the default user prompt.
temperature	A numeric value controlling the randomness of the output (default: 0.5).
api_key	API key for OpenAI or Gemini. If NULL, uses environment variable.
openai_api_key	Deprecated. Use api_key instead. Kept for backward compatibility.
verbose	Logical, if TRUE, prints progress messages.

Value

A data frame containing the top terms for each topic along with their generated labels.

See Also

get_topic_terms() to extract top terms first; generate_topic_content() for survey items / RQs / themes grounded in the same terms; call_llm_api() for the direct AI provider call

Examples

```
if (interactive()) {
top_topic_terms <- get_topic_terms(stm_model, top_term_n = 10)

# Auto-detect provider (tries OpenAI -> Gemini)
labels <- generate_topic_labels(top_topic_terms)

# Use specific provider
labels_openai <- generate_topic_labels(top_topic_terms, provider = "openai")
labels_gemini <- generate_topic_labels(top_topic_terms, provider = "gemini")
}
```

---

get_best_embeddings *Get Best Available Embeddings*

Description

Auto-detects and uses the best available embedding provider with the following priority:

1. sentence-transformers (local Python) - if Python environment is set up
2. OpenAI API - if OPENAI_API_KEY is set
3. Gemini API - if GEMINI_API_KEY is set

Usage

```
get_best_embeddings(
  texts,
  provider = "auto",
  model = NULL,
  api_key = NULL,
  verbose = TRUE
)
```

Arguments

texts	Character vector of texts to embed
provider	Character string: "auto" (default), "sentence-transformers", "openai", or "gemini". Use "auto" for automatic detection.
model	Character string specifying the embedding model. If NULL, uses default model for the selected provider.
api_key	Optional API key for OpenAI or Gemini providers. If NULL, falls back to environment variables (OPENAI_API_KEY, GEMINI_API_KEY).
verbose	Logical, whether to print progress messages (default: TRUE)

Value

Matrix with embeddings (rows = texts, columns = dimensions)

Examples

```
if (interactive()) {
data(SpecialEduTech)
texts <- SpecialEduTech$abstract[1:5]

# Auto-detect best available provider
embeddings <- get_best_embeddings(texts)

# Force specific provider
embeddings <- get_best_embeddings(texts, provider = "openai")

dim(embeddings)
}
```

---

get_content_type_prompt

Get Default System Prompt for Content Type

Description

Returns the default system prompt for a given content type.

Usage

```
get_content_type_prompt(content_type)
```

Arguments

content_type	Type of content to generate.

Value

Character string with the system prompt.

---

get_content_type_user_template *Get Default User Prompt Template for Content Type*

Description

Returns the default user prompt template for a given content type.

Usage

```
get_content_type_user_template(content_type)
```

Arguments

content_type	Type of content to generate.

Value

Character string with the user prompt template containing `{terms}` placeholder.

---

get_sentences *Get Sentences*

Description

Segment texts into sentences using spaCy's sentence boundary detection.

Usage

```
get_sentences(x, model = "en_core_web_sm")
```

Arguments

x	Character vector of texts OR a quanteda tokens object.
model	Character; spaCy model to use (default: "en_core_web_sm").

Value

A data frame with sentence information.

`get_spacy_model_info` *Get spaCy Model Information*

Description

Get information about the currently loaded spaCy model.

Usage

```
get_spacy_model_info()
```

Value

A list with model information including name, language, pipeline components, and vector availability.

`get_topic_prevalence` *Get Topic Prevalence (Gamma) from STM Model*

Description

Extracts topic prevalence values (gamma/theta) from a fitted STM model, returning mean prevalence for each topic as a data frame.

Usage

```
get_topic_prevalence(stm_model, category = NULL, include_theta = FALSE)
```

Arguments

`stm_model`	A fitted STM model object from stm::stm().
`category`	Optional character string to add as a category column.
`include_theta`	Logical, if TRUE includes document-topic matrix (default: FALSE).

Value

A data frame with columns:

`topic`	Topic number
`gamma`	Mean topic prevalence across documents
`category`	Category label (if provided)

Examples

```
if (interactive() && requireNamespace("stm", quietly = TRUE)) {
  # Requires fitting an STM model first; uses 'stm::gadarian' for demo
  data("gadarian", package = "stm")
  proc <- stm::textProcessor(gadarian$open.ended.response, metadata = gadarian)
  prep <- stm::prepDocuments(proc$documents, proc$vocab, proc$meta)
  topic_model <- stm::stm(prep$documents, prep$vocab, K = 3,
                          data = prep$meta, max.em.its = 5,
                          verbose = FALSE)
  prevalence <- get_topic_prevalence(topic_model)
  prevalence_label <- get_topic_prevalence(topic_model, category = "demo")
}
```

---

get_topic_terms *Select Top Terms for Each Topic*

Description

This function selects the top terms for each topic based on their word probability distribution (beta).

Usage

```
get_topic_terms(stm_model, top_term_n = 10, verbose = TRUE, ...)
```

Arguments

stm_model	An STM model object.
top_term_n	The number of top terms to display for each topic (default: 10).
verbose	Logical, if TRUE, prints progress messages.
...	Further arguments passed to tidytext::tidy.

Value

A data frame containing the top terms for each topic.

Examples

```
if (interactive() && requireNamespace("stm", quietly = TRUE)) {
  mydata <- TextAnalysisR::SpecialEduTech

  united_tbl <- TextAnalysisR::unite_cols(
    mydata,
    listed_vars = c("title", "keyword", "abstract")
  )

  tokens <- TextAnalysisR::prep_texts(united_tbl, text_field = "united_texts")

  dfm_object <- quanteda::dfm(tokens)
```

```
  out <- quanteda::convert(dfm_object, to = "stm")

  stm_15 <- stm::stm(
    data = out$meta,
    documents = out$documents,
    vocab = out$vocab,
    max.em.its = 75,
    init.type = "Spectral",
    K = 15,
    prevalence = ~ reference_type + s(year),
    verbose = TRUE
  )

  top_topic_terms <- TextAnalysisR::get_topic_terms(
    stm_model = stm_15,
    top_term_n = 10,
    verbose = TRUE
  )
  print(top_topic_terms)
}
```

`get_topic_texts` *Convert Topic Terms to Text Strings*

Description

Concatenates top terms for each topic into text strings suitable for embedding generation. Useful for creating topic representations for semantic similarity analysis.

Usage

```
get_topic_texts(
  top_terms_df,
  topic_var = "topic",
  term_var = "term",
  weight_var = NULL,
  sep = " ",
  top_n = NULL
)
```

Arguments

`top_terms_df`	A data frame containing top terms for topics, typically output from `get_topic_terms`.
`topic_var`	Name of the column containing topic identifiers (default: "topic").
`term_var`	Name of the column containing terms (default: "term").
`weight_var`	Optional name of column with term weights (e.g., "beta"). If provided, terms are ordered by weight before concatenation.

sep	Separator between terms (default: " ").
top_n	Optional number of top terms to include per topic (default: NULL, uses all).

Value

A character vector of topic text strings, one per topic, ordered by topic number.

Examples

```
# Topic-term frame as produced by get_topic_terms()
top_terms <- data.frame(
  topic = c(1, 1, 1, 2, 2, 2),
  term  = c("calculator", "arithmetic", "elementary",
            "computer", "instruction", "multiplication"),
  prob  = c(0.10, 0.08, 0.07, 0.12, 0.09, 0.06)
)
get_topic_texts(top_terms)
get_topic_texts(top_terms, weight_var = "prob", top_n = 2)
```

---

get_word_similarity *Calculate Word Similarity*

Description

Calculate semantic similarity between two words using word vectors. Requires a spaCy model with word vectors (en_core_web_md or en_core_web_lg).

Usage

```
get_word_similarity(word1, word2, model = "en_core_web_md")
```

Arguments

word1	Character; first word.
word2	Character; second word.
model	Character; spaCy model to use (default: "en_core_web_md").

Value

A list with similarity score and metadata.

identify_topic_trends *Identify Topic Trends*

Description

Identifies trending topics in temporal results

Usage

```
identify_topic_trends(temporal_results, ...)
```

Arguments

temporal_results	Temporal analysis results
...	Additional parameters

Value

List containing identified trends

import_files *Process Files*

Description

This function processes different types of files and text input based on the dataset choice.

Usage

```
import_files(dataset_choice, file_info = NULL, text_input = NULL)
```

Arguments

dataset_choice	A character string indicating the dataset choice.
file_info	A data frame containing file information with a column named 'filepath' (default: NULL).
text_input	A character string containing text input (default: NULL).

Value

A data frame containing the processed data.

Examples

```
if (interactive()) {
  mydata <- TextAnalysisR::SpecialEduTech
  mydata <- TextAnalysisR::import_files(dataset_choice = "Upload an Example Dataset")
  head(mydata)

  xlsx_path <- system.file("extdata", "SpecialEduTech.xlsx",
                           package = "TextAnalysisR")
  file_info <- data.frame(filepath = xlsx_path)
  mydata <- TextAnalysisR::import_files(dataset_choice = "Upload Your File",
                                        file_info = file_info)
  head(mydata)


  text_input <- paste("Virtual manipulatives for algebra instruction",
                      "manipulatives mathematics learning disability",
                      "This study examined virtual manipulatives effects on",
                      "students with learning disabilities")
  mydata <- TextAnalysisR::import_files(dataset_choice = "Copy and Paste Text",
                                        text_input = text_input)
  head(mydata)
}
```

---

lemmatize_tokens *Lemmatize Tokens with Batch Processing*

Description

Converts tokens to their lemmatized forms using spaCy, with batch processing to handle large document collections without timeout issues.

Usage

```
lemmatize_tokens(
  tokens,
  batch_size = 50,
  model = "en_core_web_sm",
  verbose = TRUE
)
```

Arguments

tokens	A quanteda tokens object to lemmatize.
batch_size	Integer; number of documents to process per batch (default: 50).
model	Character; spaCy model to use (default: "en_core_web_sm").
verbose	Logical; print progress messages (default: TRUE).

Details

Uses spaCy for linguistic lemmatization producing proper dictionary forms (e.g., "studies" -> "study", "better" -> "good"). Batch processing prevents timeout errors with large document collections.

Value

A quanteda tokens object containing lemmatized tokens.

Examples

```
if (interactive()) {
tokens <- quanteda::tokens(c("The studies showed better results"))
lemmatized <- lemmatize_tokens(tokens, batch_size = 50)
}
```

---

lexical_analysis *Lexical Analysis Functions*

Description

Functions for lexical analysis including:

- Linguistic Annotation (POS tagging, NER)
- Frequency Analysis (word frequency, n-grams, MWEs)
- Keywords (TF-IDF, keyness)
- Lexical Diversity (TTR, MTLD, MATTR)
- Readability (Flesch, Gunning Fog, etc.)

---

lexical_diversity_analysis *Lexical Diversity Analysis*

Description

Calculates multiple lexical diversity metrics for a document-feature matrix (DFM) or tokens object. Supports all quanteda.textstats measures plus MTLD (Measure of Textual Lexical Diversity), which is the most recommended measure according to McCarthy & Jarvis (2010) for being independent of text length.

Usage

```
lexical_diversity_analysis(x, measures = "all", texts = NULL, cache_key = NULL)
```

Arguments

x	A quanteda DFM or tokens object. Tokens object is preferred for accurate MTLD calculation since it preserves token order.
measures	Character vector of measures to calculate. Default is "all" which includes: TTR, C, R, CTTR, U, S, K, I, D, Vm, Maas, MATTR, MSTTR, MTLD, and HDD. Most recommended: "MTLD", "MATTR", or "HDD" for length-independent measures.
texts	Optional character vector of original texts. Required for MTLD calculation when using DFM input (since DFM loses token order).
cache_key	Optional cache key (e.g., from digest::digest) for caching expensive calculations. Use the same cache_key to retrieve cached results.

Details

MTLD (Measure of Textual Lexical Diversity) is calculated using the algorithm from McCarthy & Jarvis (2010). It counts the number of "factors" needed to reduce TTR below 0.72, then divides the number of tokens by the number of factors. This provides a length-independent measure of lexical diversity.

Important notes:

- For MTLD accuracy, pass a tokens object (not DFM) as input
- If using DFM, provide the 'texts' parameter for MTLD calculation
- MATTR and MSTTR window sizes are automatically adjusted for short documents
- Raw TTR falls mechanically as documents lengthen; compare TTR only across documents of similar length
- MTLD and MATTR are most reliable at 100+ tokens per document
- Results are cached when cache_key is provided for repeated analysis

Value

A list containing:

- `lexical_diversity`: Data frame with per-document lexical diversity scores
- `summary_stats`: List of summary statistics (mean, median, sd) for each measure

References

McCarthy, P. M., & Jarvis, S. (2010). MTLD, vocd-D, and HD-D: A validation study of sophisticated approaches to lexical diversity assessment. Behavior Research Methods, 42(2), 381-392.

See Also

`calculate_text_readability()` for grade-level / Flesch metrics on the same input; `calculate_lexical_dispersion()` for term spread across documents; `plot_lexical_diversity_distribution()` to visualize

Examples

```
data(SpecialEduTech)
texts <- SpecialEduTech$abstract[1:10]
corp <- quanteda::corpus(texts)
toks <- quanteda::tokens(corp)
# Preferred: pass tokens object for accurate MTLD
lex_div <- lexical_diversity_analysis(toks, texts = texts)
# With caching for repeated analysis
cache_key <- digest::digest(texts)
lex_div <- lexical_diversity_analysis(toks, texts = texts, cache_key = cache_key)
# Alternative: pass DFM with texts for MTLD accuracy
dfm_obj <- quanteda::dfm(toks)
lex_div <- lexical_diversity_analysis(dfm_obj, texts = texts)
print(lex_div)
```

---

lexical_frequency_analysis *Lexical Frequency Analysis*

Description

Wrapper function for plot_word_frequency for lexical analysis.

Usage

```
lexical_frequency_analysis(...)
```

Arguments

...	Arguments passed to plot_word_frequency

Value

A plotly bar chart of word frequencies

---

plot_cluster_terms *Plot Cluster Top Terms*

Description

Creates a horizontal bar plot showing the top terms in a cluster or document group.

Usage

```
plot_cluster_terms(
  terms,
  cluster_id = NULL,
  title = NULL,
  n_terms = 10,
  color = "#337ab7",
  height = 500,
  width = NULL
)
```

Arguments

terms	Named numeric vector of term frequencies, or data frame with 'term' and 'frequency' columns
cluster_id	Cluster identifier for the title (default: NULL)
title	Custom title (default: NULL, auto-generated from cluster_id)
n_terms	Number of top terms to display (default: 10)
color	Bar color (default: "#337ab7")
height	Plot height in pixels (default: 500)
width	Plot width in pixels (default: NULL for auto)

Value

A plotly object

---

plot_cross_category_heatmap

Plot Cross-Category Similarity Comparison

Description

Creates a faceted ggplot heatmap for cross-category document similarity comparison. Accepts either a pre-built long-format data frame or extracts from a similarity matrix.

Usage

```
plot_cross_category_heatmap(
  similarity_data,
  docs_data = NULL,
  row_var = "ld_doc_name",
  col_var = "other_doc_name",
  value_var = "cosine_similarity",
  category_var = "other_category",
  row_category = NULL,
```

```
  col_categories = NULL,
  row_display_var = NULL,
  col_display_var = NULL,
  method_name = "Cosine",
  title = NULL,
  show_values = TRUE,
  row_label = "Documents",
  label_max_chars = 25,
  order_by_numeric = TRUE,
  height = 600,
  width = NULL
)
```

Arguments

similarity_data	Either a similarity matrix (square numeric matrix) or a data frame in long format with columns for row labels, column labels, similarity values, and category.
docs_data	Data frame with document metadata (required if similarity_data is a matrix)
row_var	Column name for row document labels (default: "ld_doc_name")
col_var	Column name for column document labels (default: "other_doc_name")
value_var	Column name for similarity values (default: "cosine_similarity")
category_var	Column name for category in long-format data or docs_data (default: "other_category")
row_category	Category for row documents (used with matrix input)
col_categories	Categories for column documents (used with matrix input)
row_display_var	Column name for row display labels in tooltip (default: NULL, uses row_var)
col_display_var	Column name for column display labels in tooltip (default: NULL, uses col_var)
method_name	Similarity method name for legend (default: "Cosine")
title	Plot title (default: NULL)
show_values	Logical; show similarity values as text on tiles (default: TRUE)
row_label	Label for y-axis (default: "Documents")
label_max_chars	Maximum characters for axis labels before truncation (default: 25)
order_by_numeric	Logical; order by numeric ID extracted from labels (default: TRUE)
height	Plot height (default: 600)
width	Plot width (default: NULL)

Value

A ggplot object

Examples

```
articles <- TextAnalysisR::SpecialEduTech[1:7, ]
term_matrix <- as.matrix(quanteda::dfm(quanteda::tokens(articles$abstract)))
normalized_matrix <- term_matrix / sqrt(rowSums(term_matrix ^ 2))
similarity_matrix <- normalized_matrix %*% t(normalized_matrix)
thesis_rows <- which(articles$reference_type == "thesis")[1:3]
article_cols <- which(articles$reference_type == "journal_article")[1:4]
similarity_data <- expand.grid(
  ld_doc_name    = paste("Thesis", seq_along(thesis_rows)),
  other_doc_name = paste("Article", seq_along(article_cols)),
  stringsAsFactors = FALSE
)
similarity_data$cosine_similarity <- as.vector(
  similarity_matrix[thesis_rows, article_cols]
)
similarity_data$other_category <- articles$reference_type[article_cols]
plot_cross_category_heatmap(
  similarity_data = similarity_data,
  row_var = "ld_doc_name",
  col_var = "other_doc_name",
  value_var = "cosine_similarity",
  category_var = "other_category",
  row_label = "Theses"
)
```

---

plot_document_sentiment_trajectory

Plot Document Sentiment Trajectory

Description

Creates a line chart showing sentiment scores across documents with color gradient.

Usage

```
plot_document_sentiment_trajectory(
  sentiment_data,
  top_n = NULL,
  doc_ids = NULL,
  text_preview = NULL,
  title = "Document Sentiment Scores"
)
```

Arguments

sentiment_data	Data frame from analyze_sentiment() with sentiment_score column
top_n	Number of documents to display (default: NULL for all)

doc_ids	Optional vector of custom document IDs for display (default: NULL)
text_preview	Optional vector of text snippets for tooltips (default: NULL)
title	Plot title (default: "Document Sentiment Scores")

Value

A ggplot2 line chart with color gradient

---

plot_emotion_radar *Plot Emotion Radar Chart*

Description

Creates a polar/radar chart for NRC emotion analysis with optional grouping.

Usage

```
plot_emotion_radar(
  emotion_data,
  group_var = NULL,
  normalize = FALSE,
  title = "Emotion Analysis"
)
```

Arguments

emotion_data	Data frame with emotion scores (columns: emotion, total_score)
group_var	Optional grouping variable column name for overlaid radars (default: NULL)
normalize	Logical, whether to normalize scores to 0-100 scale (default: FALSE)
title	Plot title (default: "Emotion Analysis")

Value

A ggplot2 polar chart

plot_entity_frequencies *Plot Named Entity Frequencies*

Description

Creates a bar plot showing the frequency distribution of named entity types.

Usage

```
plot_entity_frequencies(
  entity_data,
  top_n = 20,
  title = "Named Entity Type Frequency",
  color = NULL,
  height = 500,
  width = NULL,
  custom_colors = NULL
)
```

Arguments

entity_data	Data frame containing entity data with columns: • entity: Named entity type (e.g., "PERSON", "ORG", "GPE") • n: (optional) Pre-computed frequency count If n is not present, frequencies will be computed from the data.
top_n	Number of top entity types to display (default: 20)
title	Plot title (default: "Named Entity Type Frequency")
color	Bar color (default: "#10B981")
height	Plot height in pixels (default: 500)
width	Plot width in pixels (default: NULL for auto)
custom_colors	Named vector of custom entity type colors (e.g., c(CONCEPT = "#00acc1", THEME = "#7c4dff")). Custom colors override defaults.

Value

A plotly object

Examples

```
if (interactive()) {
  entity_df <- data.frame(
    entity = c("PERSON", "ORG", "GPE", "DATE", "MONEY"),
    n = c(300, 250, 200, 150, 100)
  )
```

```
  plot_entity_frequencies(entity_df)

  # With custom colors
  plot_entity_frequencies(entity_df, custom_colors = c(PERSON = "#ff0000"))
}
```

`plot_keyness_keywords` *Plot Statistical Keyness*

Description

Creates a horizontal bar plot of distinctive keywords by keyness score.

Usage

```
plot_keyness_keywords(keyness_data, title = NULL, group_label = NULL)
```

Arguments

`keyness_data`	Data frame from extract_keywords_keyness()
`title`	Plot title (default: "Top Keywords by Keyness (G-squared)")
`group_label`	Optional label for the target group (default: NULL)

Value

A plotly bar chart

`plot_keyword_comparison`
Plot Keyword Comparison (TF-IDF vs Frequency)

Description

Creates a grouped bar plot comparing TF-IDF scores with term frequencies.

Usage

```
plot_keyword_comparison(
  tfidf_data,
  top_n = 10,
  title = NULL,
  normalized = FALSE
)
```

Arguments

tfidf_data	Data frame from extract_keywords_tfidf()
top_n	Number of keywords to display (default: 10)
title	Plot title (default: auto-generated)
normalized	Logical, whether TF-IDF scores are normalized (default: FALSE)

Value

A plotly grouped bar chart

---

plot_lexical_dispersion *Plot Lexical Dispersion*

Description

Creates an X-ray plot showing where terms appear across documents. Each row represents a term, and marks indicate occurrences.

Usage

```
plot_lexical_dispersion(
  dispersion_data,
  scale = "relative",
  title = "Lexical Dispersion",
  colors = NULL,
  height = 400,
  width = NULL,
  marker_size = 8
)
```

Arguments

dispersion_data	Data frame from calculate_lexical_dispersion()
scale	Character, "relative" or "absolute" (must match calculation)
title	Plot title (default: "Lexical Dispersion")
colors	Named vector of colors for each term, or NULL for auto
height	Plot height in pixels (default: 400)
width	Plot width in pixels (default: NULL for auto)
marker_size	Size of position markers (default: 8)

Value

A plotly object

Examples

```
tokens <- quanteda::tokens(TextAnalysisR::SpecialEduTech$abstract[1:5])
dispersion <- calculate_lexical_dispersion(tokens, c("learning", "instruction"))
plot_lexical_dispersion(dispersion)
```

---

plot_lexical_diversity_distribution

Plot Lexical Diversity Distribution

Description

Creates a boxplot showing the distribution of a lexical diversity metric.

Usage

```
plot_lexical_diversity_distribution(lexdiv_data, metric, title = NULL)
```

Arguments

lexdiv_data	Data frame from lexical_diversity_analysis()
metric	Metric to plot. Recommended: "MTLD" or "MATTR" (text-length independent)
title	Plot title (default: auto-generated)

Value

A plotly boxplot

Examples

```
abstracts <- TextAnalysisR::SpecialEduTech$abstract[1:10]
tokens <- quanteda::tokens(quanteda::corpus(abstracts))
diversity_result <- lexical_diversity_analysis(tokens, texts = abstracts)
diversity_plot <- plot_lexical_diversity_distribution(
  diversity_result$lexical_diversity, "MTLD"
)
print(diversity_plot)
```

plot_log_odds_ratio *Plot Log Odds Ratio*

Description

Creates a horizontal bar plot showing log odds ratios for comparing word usage between categories. Positive values indicate higher usage in the first category, negative in the second.

Usage

```
plot_log_odds_ratio(
  log_odds_data,
  top_n = 10,
  facet_by = NULL,
  color_positive = "#10B981",
  color_negative = "#EF4444",
  height = 600,
  width = NULL,
  title = "Log Odds Ratio Comparison"
)
```

Arguments

log_odds_data	Data frame from calculate_log_odds_ratio()
top_n	Number of top terms to show per direction (default: 10)
facet_by	Character, column name to facet by (e.g., "category1" for one_vs_rest comparisons). NULL for no faceting.
color_positive	Color for positive log odds (default: "#10B981" green)
color_negative	Color for negative log odds (default: "#EF4444" red)
height	Plot height in pixels (default: 600)
width	Plot width in pixels (default: NULL for auto)
title	Plot title (default: "Log Odds Ratio Comparison")

Value

A plotly object

Examples

```
articles <- TextAnalysisR::SpecialEduTech[1:20, ]
corpus <- quanteda::corpus(
  articles$abstract,
  docvars = data.frame(reference_type = articles$reference_type)
)
dfm_object <- quanteda::dfm(quanteda::tokens(corpus))
log_odds <- calculate_log_odds_ratio(dfm_object, "reference_type",
```

```
                                        comparison_mode = "binary")
plot_log_odds_ratio(log_odds, top_n = 5)
```

`plot_model_comparison` *Plot Topic Model Comparison Scatter*

Description

Creates a scatter plot comparing topic model metrics across K values. Automatically selects the best available metric combination.

Usage

```
plot_model_comparison(search_results, title = NULL, height = 600, width = 800)
```

Arguments

`search_results`	Results from stm::searchK or find_optimal_k()
`title`	Plot title (default `NULL` selects an auto-title based on which metric columns are present in `search_results`)
`height`	Plot height in pixels (default: 600)
`width`	Plot width in pixels (default: 800)

Value

A plotly scatter plot

`plot_morphology_feature`
Plot Morphology Feature Distribution

Description

Creates a bar chart showing the distribution of a morphological feature using consistent package styling.

Usage

```
plot_morphology_feature(data, feature, title = NULL, colors = NULL)
```

Arguments

data	Data frame with morph_* columns from extract_morphology().
feature	Character; feature name (e.g., "Number", "Tense").
title	Character; plot title (auto-generated if NULL).
colors	Named character vector of custom colors for feature values.

Value

A plotly object.

Examples

```
if (interactive()) {
  tokens <- quanteda::tokens(TextAnalysisR::SpecialEduTech$abstract[1])
  morphology_data <- extract_morphology(tokens)
  plot_morphology_feature(morphology_data, "Tense")
}
```

---

plot_mwe_frequency *Plot Multi-Word Expression Frequency*

Description

Creates a bar plot showing multi-word expression frequencies with optional source-based coloring to distinguish between detected and manually added expressions.

Usage

```
plot_mwe_frequency(
  mwe_data,
  title = "Multi-Word Expression Frequency",
  color_by_source = TRUE,
  primary_color = "#10B981",
  secondary_color = "#A855F7",
  height = 500,
  width = NULL
)
```

Arguments

mwe_data	Data frame containing MWE data with columns: • feature: The multi-word expression text • frequency: Frequency count • rank: (optional) Rank of the expression • docfreq: (optional) Document frequency • source: (optional) Source category (e.g., "Top 20", "Manual")

`title`	Plot title (default: "Multi-Word Expression Frequency")
`color_by_source`	Whether to color bars by source column (default: TRUE)
`primary_color`	Color for primary/top expressions (default: "#10B981")
`secondary_color`	Color for secondary/manual expressions (default: "#A855F7")
`height`	Plot height in pixels (default: 500)
`width`	Plot width in pixels (default: NULL for auto)

Value

A plotly object

`plot_ngram_frequency` *Plot N-gram Frequency*

Description

Creates a bar plot showing n-gram frequencies with optional highlighting of selected n-grams. Supports both detected n-grams and selected multi-word expressions.

Usage

```
plot_ngram_frequency(
  ngram_data,
  top_n = 30,
  selected = NULL,
  title = "N-gram Frequency",
  highlight_color = "#10B981",
  default_color = "#6B7280",
  height = 500,
  width = NULL,
  show_stats = TRUE
)
```

Arguments

`ngram_data`	Data frame containing n-gram data with columns: • `collocation`: The n-gram text • `count`: Frequency count • `lambda`: (optional) Lambda statistic • `z`: (optional) Z-score statistic
`top_n`	Number of top n-grams to display (default: 30)
`selected`	Character vector of selected n-grams to highlight (default: NULL)

title	Plot title (default: "N-gram Frequency")
highlight_color	Color for highlighted bars (default: "#10B981")
default_color	Color for non-highlighted bars (default: "#6B7280")
height	Plot height in pixels (default: 500)
width	Plot width in pixels (default: NULL for auto)
show_stats	Whether to show lambda and z-score in hover (default: TRUE)

Value

A plotly object

Examples

```
ngram_df <- data.frame(
  collocation = c("machine learning", "deep learning", "neural network"),
  count = c(150, 120, 90),
  lambda = c(5.2, 4.8, 4.1),
  z = c(12.3, 10.5, 9.2)
)
plot_ngram_frequency(ngram_df, selected = c("machine learning"))
```

---

plot_pos_frequencies *Plot Part-of-Speech Tag Frequencies*

Description

Creates a bar plot showing the frequency distribution of part-of-speech tags.

Usage

```
plot_pos_frequencies(
  pos_data,
  top_n = 20,
  title = "Part-of-Speech Tag Frequency",
  color = "#337ab7",
  height = 500,
  width = NULL
)
```

Arguments

pos_data	Data frame containing POS data with columns: • pos: Part-of-speech tag • n: (optional) Pre-computed frequency count If n is not present, frequencies will be computed from the data.
top_n	Number of top POS tags to display (default: 20)
title	Plot title (default: "Part-of-Speech Tag Frequency")
color	Bar color (default: "#337ab7")
height	Plot height in pixels (default: 500)
width	Plot width in pixels (default: NULL for auto)

Value

A plotly object

Examples

```
if (interactive()) {
  pos_df <- data.frame(
    pos = c("NOUN", "VERB", "ADJ", "ADV", "PRON"),
    n = c(500, 400, 250, 150, 100)
  )
  plot_pos_frequencies(pos_df)
}
```

---

plot_quality_metrics *Plot Topic Model Quality Metrics*

Description

Creates individual diagnostic metric plots across different K values from stm::searchK results.

Usage

```
plot_quality_metrics(search_results)
```

Arguments

search_results	Results from stm::searchK or find_optimal_k()

Value

A named list of ggplot objects, one per available metric (possible keys: semcoh, residual, heldout, lbound).

plot_readability_by_group *Plot Readability by Group*

Description

Creates grouped boxplots comparing readability across categories.

Usage

```
plot_readability_by_group(readability_data, metric, group_var, title = NULL)
```

Arguments

readability_data	Data frame from calculate_text_readability()
metric	Metric to plot
group_var	Name of grouping variable column
title	Plot title (default: auto-generated)

Value

A plotly boxplot

plot_readability_distribution *Plot Readability Distribution*

Description

Creates a boxplot showing the overall distribution of a readability metric.

Usage

```
plot_readability_distribution(readability_data, metric, title = NULL)
```

Arguments

readability_data	Data frame from calculate_text_readability()
metric	Metric to plot (e.g., "flesch", "flesch_kincaid", "gunning_fog")
title	Plot title (default: auto-generated)

Value

A plotly boxplot

Examples

```
data(SpecialEduTech, package = "TextAnalysisR")
texts <- SpecialEduTech$abstract[1:20]
readability <- calculate_text_readability(texts)
plot <- plot_readability_distribution(readability, "flesch")
print(plot)
```

---

plot_semantic_viz *Plot Semantic Analysis Visualization*

Description

Creates interactive visualizations for semantic analysis results including similarity heatmaps, dimensionality reduction plots, and clustering visualizations.

Usage

```
plot_semantic_viz(
  analysis_result = NULL,
  plot_type = "similarity",
  data_labels = NULL,
  color_by = NULL,
  height = 600,
  width = 800,
  title = NULL,
  coords = NULL,
  clusters = NULL,
  hover_text = NULL,
  hover_config = NULL,
  cluster_colors = NULL
)
```

Arguments

analysis_result	A list containing semantic analysis results from functions like semantic_similarity_analysis(), semantic_document_clustering(), or reduce_dimensions().
plot_type	Type of visualization: "similarity" for heatmap, "dimensionality_reduction" for scatter plot, or "clustering" for cluster visualization (default: "similarity").
data_labels	Optional character vector of labels for data points (default: NULL).
color_by	Optional variable to color points by in scatter plots (default: NULL).

height	The height of the resulting Plotly plot, in pixels (default: 600).
width	The width of the resulting Plotly plot, in pixels (default: 800).
title	Optional custom title for the plot (default: NULL).
coords	Optional pre-computed coordinates for dimensionality reduction plots (default: NULL).
clusters	Optional cluster assignments vector (default: NULL).
hover_text	Optional custom hover text for points (default: NULL).
hover_config	Optional hover configuration list (default: NULL).
cluster_colors	Optional color palette for clusters (default: NULL).

Value

A ggplot2 object showing the specified visualization.

Examples

```
data(SpecialEduTech)
texts <- SpecialEduTech$abstract[1:5]
result <- semantic_similarity_analysis(texts)
plot <- plot_semantic_viz(result, plot_type = "similarity")
print(plot)
```

---

plot_sentiment_boxplot

Plot Sentiment Box Plot by Category

Description

Creates a box plot showing sentiment score distribution by category.

Usage

```
plot_sentiment_boxplot(
  sentiment_data,
  category_var = "category_var",
  title = "Sentiment Score Distribution"
)
```

Arguments

sentiment_data	Data frame from analyze_sentiment() containing sentiment_score and category columns
category_var	Name of the category variable column (default: "category_var")
title	Plot title (default: "Sentiment Score Distribution")

Value

A ggplot2 box plot

`plot_sentiment_by_category` *Plot Sentiment by Category*

Description

Creates a grouped or stacked bar plot showing sentiment distribution across categories.

Usage

```
plot_sentiment_by_category(
  sentiment_data,
  category_var,
  plot_type = "bar",
  title = NULL
)
```

Arguments

`sentiment_data`	Data frame with 'sentiment' column
`category_var`	Name of the category variable column
`plot_type`	Type of plot: "bar" or "stacked" (default: "bar")
`title`	Plot title (default: auto-generated)

Value

A ggplot2 grouped/stacked bar chart

Examples

```
articles <- TextAnalysisR::SpecialEduTech[1:20, ]
sentiment_results <- analyze_sentiment(articles$abstract)
sentiment_data <- cbind(
  reference_type = articles$reference_type,
  sentiment_results
)
sentiment_plot <- plot_sentiment_by_category(sentiment_data, "reference_type")
print(sentiment_plot)
```

plot_sentiment_distribution *Plot Sentiment Distribution*

Description

Creates a bar plot showing the distribution of sentiment classifications.

Usage

```
plot_sentiment_distribution(sentiment_data, title = "Sentiment Distribution")
```

Arguments

sentiment_data	Data frame from analyze_sentiment() or with 'sentiment' column
title	Plot title (default: "Sentiment Distribution")

Value

A ggplot2 bar chart

Examples

```
abstracts <- TextAnalysisR::SpecialEduTech$abstract[1:10]
sentiment_data <- analyze_sentiment(abstracts)
sentiment_plot <- plot_sentiment_distribution(sentiment_data)
print(sentiment_plot)
```

plot_sentiment_violin *Plot Sentiment Violin Plot by Category*

Description

Creates a violin plot showing sentiment score distribution by category.

Usage

```
plot_sentiment_violin(
  sentiment_data,
  category_var = "category_var",
  title = "Sentiment Score Distribution"
)
```

Arguments

sentiment_data	Data frame from analyze_sentiment() containing sentiment_score and category columns
category_var	Name of the category variable column (default: "category_var")
title	Plot title (default: "Sentiment Score Distribution")

Value

A ggplot2 violin plot

---

plot_similarity_heatmap

Plot Document Similarity Heatmap

Description

Creates an interactive heatmap visualization of document similarity matrices with support for document metadata, feature-specific colorscales, and rich tooltips. Supports both symmetric (all-vs-all) and cross-category comparison modes.

Usage

```
plot_similarity_heatmap(
  similarity_matrix,
  docs_data = NULL,
  feature_type = "words",
  method_name = "Cosine",
  title = NULL,
  category_filter = NULL,
  doc_id_var = NULL,
  colorscale = NULL,
  height = 600,
  width = NULL,
  row_category = NULL,
  col_categories = NULL,
  category_var = "category_display",
  show_values = FALSE,
  facet = NULL,
  row_label = NULL,
  output_type = "plotly"
)
```

Arguments

similarity_matrix	A square numeric matrix of similarity scores
docs_data	Optional data frame with document metadata containing: • document_number: Document identifiers for axis labels • document_id_display: Document IDs for hover text • category_display: Category labels for hover text
feature_type	Feature space type: "words", "topics", "ngrams", or "embeddings" (determines colorscale and display name)
method_name	Similarity method name for display (default: "Cosine")
title	Plot title (default: NULL, auto-generated from feature_type)
category_filter	Optional category filter label for title (default: NULL)
doc_id_var	Name of document ID variable (affects label text, default: NULL)
colorscale	Plotly colorscale override (default: NULL, uses feature_type default)
height	Plot height in pixels (default: 600)
width	Plot width in pixels (default: NULL for auto)
row_category	Category for row documents in cross-category mode (default: NULL)
col_categories	Character vector of categories for column documents (default: NULL)
category_var	Name of category variable in docs_data (default: "category_display")
show_values	Logical; show similarity values as text on tiles (default: FALSE)
facet	Logical; facet by column categories (default: TRUE when col_categories specified)
row_label	Label for row axis (default: NULL, uses row_category)
output_type	Output type: "plotly" or "ggplot" (default: "plotly", auto-switches to "ggplot" for faceting)

Value

A ggplot2 heatmap object

Examples

```
articles <- TextAnalysisR::SpecialEduTech[1:5, ]
term_matrix <- as.matrix(quanteda::dfm(quanteda::tokens(articles$abstract)))
normalized_matrix <- term_matrix / sqrt(rowSums(term_matrix ^ 2))
similarity_matrix <- normalized_matrix %*% t(normalized_matrix)
plot_similarity_heatmap(similarity_matrix)

document_metadata <- data.frame(
  document_number     = paste("Doc", 1:5),
  document_id_display = articles$title,
  category_display    = articles$reference_type
)
```

```
plot_similarity_heatmap(similarity_matrix, docs_data = document_metadata,
                        feature_type = "embeddings")

plot_similarity_heatmap(
  similarity_matrix,
  docs_data      = document_metadata,
  row_category   = "thesis",
  col_categories = "journal_article",
  show_values    = TRUE,
  facet          = TRUE
)
```

`plot_term_trends_continuous` *Plot Term Frequency Trends by Continuous Variable*

Description

Creates a faceted line plot showing how term frequencies vary across a continuous variable (e.g., year, time period).

Usage

```
plot_term_trends_continuous(
  term_data,
  continuous_var,
  terms = NULL,
  title = NULL,
  height = 600,
  width = NULL
)
```

Arguments

`term_data`	Data frame containing term frequencies with columns: continuous_var, term, and word_frequency
`continuous_var`	Name of the continuous variable column
`terms`	Character vector of terms to display (optional, filters if provided)
`title`	Plot title (default: NULL, auto-generated)
`height`	Plot height in pixels (default: 600)
`width`	Plot width in pixels (default: NULL, auto)

Value

A plotly object with faceted line plots

Examples

```
term_df <- data.frame(
  year = rep(2010:2020, each = 3),
  term = rep(c("learning", "education", "technology"), 11),
  word_frequency = sample(10:100, 33, replace = TRUE)
)
plot_term_trends_continuous(term_df, "year", c("learning", "education"))
```

---

plot_tfidf_keywords *Plot TF-IDF Keywords*

Description

Creates a horizontal bar plot of top keywords by TF-IDF score.

Usage

```
plot_tfidf_keywords(tfidf_data, title = NULL, normalized = FALSE)
```

Arguments

tfidf_data	Data frame from extract_keywords_tfidf()
title	Plot title (default: "Top Keywords by TF-IDF Score")
normalized	Logical, whether scores are normalized (for label) (default: FALSE)

Value

A plotly bar chart

---

plot_topic_effects_categorical *Plot Topic Effects for Categorical Variables*

Description

Creates a faceted plot showing how categorical variables affect topic proportions.

Usage

```
plot_topic_effects_categorical(
  effects_data,
  ncol = 2,
  height = 800,
  width = 1000,
  title = "Category Effects",
  base_font_size = 11
)
```

Arguments

effects_data	Data frame with columns: topic, value, proportion, lower, upper
ncol	Number of columns for faceting (default: 2)
height	Plot height in pixels (default: 800)
width	Plot width in pixels (default: 1000)
title	Plot title (default: "Category Effects")
base_font_size	Base font size in points for the plot theme (default: 11). Axis text and strip text will be base_font_size + 2.

Value

A plotly object

---

plot_topic_effects_continuous

Plot Topic Effects for Continuous Variables

Description

Creates a faceted plot showing how continuous variables affect topic proportions.

Usage

```
plot_topic_effects_continuous(
  effects_data,
  ncol = 2,
  height = 800,
  width = 1000,
  title = "Continuous Variable Effects",
  base_font_size = 11
)
```

Arguments

effects_data	Data frame with columns: topic, value, proportion, lower, upper
ncol	Number of columns for faceting (default: 2)
height	Plot height in pixels (default: 800)
width	Plot width in pixels (default: 1000)
title	Plot title (default: "Continuous Variable Effects")
base_font_size	Base font size in points for the plot theme (default: 11). Axis text and strip text will be base_font_size + 2.

Value

A plotly object

plot_topic_probability *Plot Per-Document Per-Topic Probabilities*

Description

Generates a bar plot showing the prevalence of each topic across all documents.

Usage

```
plot_topic_probability(
  gamma_data,
  top_n = 10,
  use_topic_labels = FALSE,
  colors = NULL,
  ylab = "Topic Proportion",
  base_font_size = 11
)
```

Arguments

gamma_data	A data frame with gamma values from calculate_topic_probability().
top_n	The number of topics to display (default: 10).
use_topic_labels	Logical. If TRUE, use the topic_label column from gamma_data for axis labels (falls back to topic number when the column is absent). If FALSE (default), labels are formatted as "Topic N".
colors	Optional color palette for topics (default: NULL).
ylab	Y-axis label (default: "Topic Proportion").
base_font_size	Base font size in points for the plot theme (default: 11). Axis text and strip text will be base_font_size + 2.

Value

A ggplot2 object showing a bar plot of topic prevalence.

plot_top_readability_documents

Plot Top Documents by Readability

Description

Creates a bar plot of documents ranked by readability metric.

Usage

```
plot_top_readability_documents(
  readability_data,
  metric,
  top_n = 15,
  order = "highest",
  title = NULL
)
```

Arguments

readability_data	Data frame from calculate_text_readability()
metric	Metric to plot
top_n	Number of documents to show (default: 15)
order	Direction: "highest" or "lowest" (default: "highest")
title	Plot title (default: auto-generated)

Value

A plotly bar chart

plot_weighted_log_odds

Plot Weighted Log Odds

Description

Creates a faceted horizontal bar plot showing weighted log odds for comparing word usage across categories using the Fightin' Words method (Monroe et al. 2008). Each group is displayed in a separate facet showing its most distinctive terms.

Usage

```
plot_weighted_log_odds(
  weighted_data,
  top_n = 10,
  color_positive = "#10B981",
  color_negative = "#EF4444",
  height = 600,
  width = NULL,
  title = "Weighted Log Odds by Group"
)
```

Arguments

weighted_data	Data frame from calculate_weighted_log_odds()
top_n	Number of top terms to show per group (default: 10)
color_positive	Color for positive log odds (default: "#10B981" green)
color_negative	Color for negative log odds (default: "#EF4444" red)
height	Plot height in pixels (default: 600)
width	Plot width in pixels (default: NULL for auto)
title	Plot title (default: "Weighted Log Odds by Group")

Value

A plotly object

Examples

```
if (requireNamespace("tidylo", quietly = TRUE)) {
  articles <- TextAnalysisR::SpecialEduTech[1:20, ]
  dfm_object <- quanteda::dfm(quanteda::tokens(articles$abstract))
  quanteda::docvars(dfm_object, "reference_type") <- articles$reference_type
  weighted_odds <- calculate_weighted_log_odds(dfm_object, "reference_type",
                                               top_n = 5)
  plot_weighted_log_odds(weighted_odds)
}
```

---

plot_word_frequency *Plot Word Frequency*

Description

Creates a bar plot showing the most frequent words in a document-feature matrix (dfm).

Usage

```
plot_word_frequency(dfm_object, n = 20, height = NULL, width = NULL, ...)
```

Arguments

dfm_object	A document-feature matrix created by quanteda::dfm().
n	The number of top words to display (default: 20).
height	Plot height in pixels (default: 800). Kept for backward compatibility.
width	Plot width in pixels (default: 1000). Kept for backward compatibility.
...	Additional arguments (kept for backward compatibility).

Value

A ggplot object showing word frequency.

Examples

```
data(SpecialEduTech, package = "TextAnalysisR")
texts <- SpecialEduTech$abstract[1:10]
dfm <- quanteda::dfm(quanteda::tokens(texts))
plot <- plot_word_frequency(dfm, n = 10)
print(plot)
```

---

plot_word_probability *Plot Word Probabilities by Topic*

Description

Creates a faceted bar plot showing the top terms and their probabilities (beta values) for each topic in a topic model.

Usage

```
plot_word_probability(
  top_topic_terms,
  topic_label = NULL,
  ncol = 3,
  height = 1200,
  width = 800,
  ylab = "Word probability",
  title = NULL,
  colors = NULL,
  measure_label = "Beta",
  base_font_size = 11,
  ...
)
```

Arguments

top_topic_terms	A data frame containing topic terms with columns: topic, term, and beta.
topic_label	Optional topic labels. Can be either a named vector mapping topic numbers to labels, or a character string specifying a column name in top_topic_terms (default: NULL).
ncol	Number of columns for facet wrap layout (default: 3).
height	Plot height for responsive spacing adjustments (default: 1200).
width	Plot width for responsive spacing adjustments (default: 800).
ylab	Y-axis label (default: "Word probability").
title	Plot title (default: NULL for auto-generated title).
colors	Color palette for topics (default: NULL for auto-generated colors).
measure_label	Label for the probability measure (default: "Beta").
base_font_size	Base font size in points for the plot theme (default: 11). Axis text and strip text will be base_font_size + 2.
...	Additional arguments (currently unused, kept for compatibility).

Value

A ggplot2 object showing word probabilities faceted by topic.

---

prep_texts *Preprocess Text Data*

Description

Preprocesses text data following the complete workflow implemented in the Shiny application:

- Constructing a corpus from united texts
- Tokenizing text into words with configurable options
- Converting to lowercase with acronym preservation option
- Applying character length filtering
- Optional multi-word expression detection and compound term creation
- Stopword removal and lemmatization capabilities

This function serves as the foundation for all subsequent text analysis workflows.

Usage

```
prep_texts(
  united_tbl,
  text_field = "united_texts",
  min_char = 2,
  lowercase = TRUE,
  remove_punct = TRUE,
  remove_symbols = TRUE,
  remove_numbers = TRUE,
  remove_url = TRUE,
  remove_separators = TRUE,
  split_hyphens = TRUE,
  split_tags = TRUE,
  include_docvars = TRUE,
  keep_acronyms = FALSE,
  padding = FALSE,
  remove_stopwords = FALSE,
  stopwords_source = "snowball",
  stopwords_language = "en",
  custom_stopwords = NULL,
  custom_valuetype = "glob",
  math_mode = FALSE,
  verbose = FALSE,
  ...
)
```

Arguments

united_tbl	A data frame that contains text data.
text_field	The name of the column that contains the text data.
min_char	The minimum number of characters for a token to be included (default: 2).
lowercase	Logical; convert all tokens to lowercase (default: TRUE). Recommended for most text analysis tasks.
remove_punct	Logical; remove punctuation from the text (default: TRUE).
remove_symbols	Logical; remove symbols from the text (default: TRUE).
remove_numbers	Logical; remove numbers from the text (default: TRUE).
remove_url	Logical; remove URLs from the text (default: TRUE).
remove_separators	Logical; remove separators from the text (default: TRUE).
split_hyphens	Logical; split hyphenated words into separate tokens (default: TRUE).
split_tags	Logical; split tags into separate tokens (default: TRUE).
include_docvars	Logical; include document variables in the tokens object (default: TRUE).
keep_acronyms	Logical; keep acronyms in the text (default: FALSE).
padding	Logical; add padding to the tokens object (default: FALSE).

remove_stopwords
: Logical; remove stopwords from the text (default: FALSE).

stopwords_source
: Character; source for stopwords, e.g., "snowball", "stopwords-iso" (default: "snowball").

stopwords_language
: Character; language for stopwords (default: "en").

custom_stopwords
: Character vector; additional words to remove (default: NULL).

custom_valuetype
: Character; valuetype for custom_stopwords pattern matching, one of "glob", "regex", or "fixed" (default: "glob").

math_mode
: Logical; if TRUE, preserve math content (numbers, operators, symbols) by forcing remove_punct, remove_symbols, and remove_numbers all to FALSE, then strip only sentence-end punctuation such as periods, commas, question marks, exclamation marks, colons, semicolons, parentheses, brackets, braces, quotation marks, em dashes, and en dashes. The min_char default of 2 still applies, so noisy single-character tokens are dropped; pass min_char = 1 to keep them. Use for math or STEM corpora where multi-character operators and numerals carry meaning (default: FALSE).

verbose
: Logical; print verbose output (default: FALSE).

...
: Additional arguments passed to quanteda::tokens.

Value

A tokens object that contains the preprocessed text data.

See Also

unite_cols() to combine text columns first; lemmatize_tokens() to reduce words to base form (e.g., running -> run); quanteda::dfm() to build a document-feature matrix from the result

Examples

```
mydata <- TextAnalysisR::SpecialEduTech

united_tbl <- TextAnalysisR::unite_cols(
  mydata,
  listed_vars = c("title", "keyword", "abstract")
)

tokens <- TextAnalysisR::prep_texts(united_tbl,
                                    text_field = "united_texts",
                                    min_char = 2,
                                    lowercase = TRUE,
                                    remove_punct = TRUE,
                                    remove_symbols = TRUE,
                                    remove_numbers = TRUE,
                                    remove_url = TRUE,
```

```
                                        remove_separators = TRUE,
                                        split_hyphens = TRUE,
                                        split_tags = TRUE,
                                        include_docvars = TRUE,
                                        keep_acronyms = FALSE,
                                        padding = FALSE,
                                        verbose = FALSE)
print(tokens)
```

`process_pdf_unified` *Process PDF File (Unified Entry Point)*

Description

Unified PDF processing:

1. Multimodal (R-native pdftools + Vision LLM) if enabled
2. R pdftools text extraction as fallback

Usage

```
process_pdf_unified(
  file_path,
  use_multimodal = FALSE,
  vision_provider = "gemini",
  vision_model = NULL,
  api_key = NULL,
  describe_images = TRUE
)
```

Arguments

`file_path`	Character string path to PDF file
`use_multimodal`	Logical, enable multimodal extraction
`vision_provider`	Character, "openai" or "gemini"
`vision_model`	Character, model name
`api_key`	Character, API key (if using openai/gemini)
`describe_images`	Logical, generate image descriptions

Value

List: success, data, type, method, message

See Also

`import_files()` for the generic file-loading entry point; `describe_image()` for standalone image captioning

---

`reduce_dimensions` *Dimensionality Reduction Analysis*

Description

This function performs dimensionality reduction using various methods including PCA, t-SNE, and UMAP. UMAP runs directly on inputs with 1000 or fewer features (e.g. dense embeddings); PCA pre-reduction applies to wider inputs and to the PCA and t-SNE methods.

Usage

```
reduce_dimensions(
  data_matrix,
  method = "PCA",
  n_components = 2,
  pca_dims = 50,
  tsne_perplexity = 30,
  tsne_max_iter = 1000,
  umap_neighbors = 15,
  umap_min_dist = 0.1,
  umap_metric = "cosine",
  seed = 123,
  verbose = TRUE
)
```

Arguments

`data_matrix`	A numeric matrix where rows represent documents and columns represent features.
`method`	The dimensionality reduction method. Options: "PCA", "t-SNE", "UMAP".
`n_components`	The number of components/dimensions to reduce to (default: 2).
`pca_dims`	The number of dimensions for PCA preprocessing (default: 50).
`tsne_perplexity`	The perplexity parameter for t-SNE (default: 30).
`tsne_max_iter`	The maximum number of iterations for t-SNE (default: 1000).
`umap_neighbors`	The number of neighbors for UMAP (default: 15).
`umap_min_dist`	The minimum distance for UMAP (default: 0.1).
`umap_metric`	The metric for UMAP (default: "cosine").
`seed`	Random seed for reproducibility (default: 123).
`verbose`	Logical, if TRUE, prints progress messages.

Value

A list containing the reduced dimensions, method used, and additional metadata.

Examples

```
if (interactive()) {
  mydata <- TextAnalysisR::SpecialEduTech

  united_tbl <- TextAnalysisR::unite_cols(
    mydata,
    listed_vars = c("title", "keyword", "abstract")
  )

  tokens <- TextAnalysisR::prep_texts(united_tbl, text_field = "united_texts")

  dfm_object <- quanteda::dfm(tokens)

  data_matrix <- as.matrix(dfm_object)

  pca_result <- TextAnalysisR::reduce_dimensions(
    data_matrix,
    method = "PCA"
  )
  print(pca_result)

  tsne_result <- TextAnalysisR::reduce_dimensions(
    data_matrix,
    method = "t-SNE"
  )
  print(tsne_result)

  umap_result <- TextAnalysisR::reduce_dimensions(
    data_matrix,
    method = "UMAP"
  )
  print(umap_result)
}
```

---

render_displacy_dep *Render displaCy Dependency Visualization*

Description

Renders spaCy's displaCy dependency visualization as SVG. Shows syntactic structure with arrows between words.

Usage

```
render_displacy_dep(text, compact = TRUE, model = "en_core_web_sm")
```

Arguments

text	Character string to visualize.
compact	Logical; use compact mode for space (default: TRUE).
model	spaCy model name (default: "en_core_web_sm").

Value

SVG string with dependency tree.

---

render_displacy_ent *Render displaCy Entity Visualization*

Description

Renders spaCy's displaCy entity visualization as HTML. Highlights named entities with colored labels.

Usage

```
render_displacy_ent(text, model = "en_core_web_sm", colors = NULL)
```

Arguments

text	Character string to visualize.
model	spaCy model name (default: "en_core_web_sm").
colors	Named list of entity type to color mappings (e.g., list(PERSON = "#e91e63", ORG = "#2196f3")). If NULL, uses spaCy defaults.

Value

HTML string with entity highlighting.

---

run_app *Launch the TextAnalysisR app*

Description

Launch the TextAnalysisR Shiny application.

Usage

```
run_app(launch.browser = interactive())
```

Arguments

launch.browser	Logical. Whether to open the app in a browser. Defaults to interactive(), which is FALSE in non-interactive sessions (e.g., Docker containers, servers).

Value

No return value, called for side effects (launching Shiny app).

Examples

```
if (interactive()) {
  run_app()
}
```

---

run_rag_search *RAG Semantic Search*

Description

Simple in-memory RAG (Retrieval Augmented Generation) for question-answering over document corpus with source attribution. Uses OpenAI or Gemini embeddings for semantic search and LLM for answer generation.

Usage

```
run_rag_search(
  query,
  documents,
  provider = c("openai", "gemini"),
  api_key = NULL,
  embedding_model = NULL,
  chat_model = NULL,
  top_k = 5
)
```

Arguments

query	Character string, user question
documents	Character vector, corpus to search
provider	Character string, provider: "openai" or "gemini"
api_key	Character string, API key (or from OPENAI_API_KEY/GEMINI_API_KEY env).
embedding_model	Character string, embedding model. Defaults: "text-embedding-3-small" (openai), "gemini-embedding-001" (gemini)

chat_model	Character string, chat model. Defaults: "gpt-4.1-mini" (openai), "gemini-2.5-flash-lite" (gemini)
top_k	Integer, number of documents to retrieve (default: 5)

Details

Simple RAG workflow:

1. Generate embeddings for documents and query
2. Find top-k similar documents via cosine similarity
3. Generate answer using LLM with retrieved context

Value

List with:

- success: Logical
- answer: Generated answer
- confidence: Confidence score (0-1)
- sources: Vector of source document indices
- retrieved_docs: Retrieved document chunks
- scores: Similarity scores

See Also

`get_best_embeddings()` for the retrieval step alone; `call_llm_api()` for the answer-generation step alone; `sanitize_llm_input()` for an input safety check before calling

Examples

```
if (interactive()) {
documents <- c(
  "Assistive technology helps students with disabilities access curriculum.",
  "Universal Design for Learning provides multiple means of engagement.",
  "Response to Intervention uses tiered support systems."
)

# Using OpenAI (requires API key)
result <- run_rag_search(
  query = "How does assistive technology support learning?",
  documents = documents,
  provider = "openai"
)

if (result$success) {
  cat("Answer:", result$answer, "\n")
  cat("Sources:", paste(result$sources, collapse = ", "), "\n")
}
}
```

`run_text_workflow` *Complete Text Mining Workflow*

Description

This function provides a complete text mining workflow that follows the same sequence as the Shiny application: file processing -> text uniting -> preprocessing -> DFM creation -> analysis. It serves as a convenience function for users who want to execute the entire pipeline programmatically.

Usage

```
run_text_workflow(
  dataset_choice,
  file_info = NULL,
  text_input = NULL,
  listed_vars,
  min_char = 2,
  remove_punct = TRUE,
  remove_symbols = TRUE,
  remove_numbers = TRUE,
  remove_url = TRUE,
  detect_compounds = FALSE,
  compound_size = 2:3,
  compound_min_count = 2,
  verbose = TRUE
)
```

Arguments

`dataset_choice`	A character string indicating the dataset choice: "Upload an Example Dataset", "Upload Your File", "Copy and Paste Text".
`file_info`	A data frame containing file information (for file upload).
`text_input`	A character string containing text input (for copy-paste).
`listed_vars`	A character vector of column names to unite into text.
`min_char`	The minimum number of characters for tokens (default: 2).
`remove_punct`	Logical; remove punctuation (default: TRUE).
`remove_symbols`	Logical; remove symbols (default: TRUE).
`remove_numbers`	Logical; remove numbers (default: TRUE).
`remove_url`	Logical; remove URLs (default: TRUE).
`detect_compounds`	Logical; detect multi-word expressions (default: FALSE).
`compound_size`	Size range for compound detection (default: 2:3).
`compound_min_count`	Minimum count for compounds (default: 2).
`verbose`	Logical; print progress messages (default: TRUE).

Value

A list containing processed data, tokens, DFM, and metadata.

Examples

```
  workflow_result <- TextAnalysisR::run_text_workflow(
    dataset_choice = "Upload an Example Dataset",
    listed_vars = c("title", "keyword", "abstract")
  )

if (interactive()) {
  file_info <- data.frame(filepath = "path/to/file.xlsx")
  workflow_result <- TextAnalysisR::run_text_workflow(
    dataset_choice = "Upload Your File",
    file_info = file_info,
    listed_vars = c("column1", "column2")
  )

  workflow_result <- TextAnalysisR::run_text_workflow(
    dataset_choice = "Copy and Paste Text",
    text_input = "Your text content here",
    listed_vars = "text"
  )
}
```

---

semantic_document_clustering

Semantic Document Clustering

Description

Creates a unified visualization of document clustering with optional clustering

Usage

```
semantic_document_clustering(embeddings, method = "UMAP", clusters = NULL, ...)
```

Arguments

embeddings	Document embeddings matrix
method	Dimensionality reduction method ("PCA", "t-SNE", "UMAP")
clusters	Optional cluster assignments
...	Additional arguments

Value

A ggplot2 visualization of document clusters

semantic_similarity_analysis

Semantic Similarity Analysis

Description

Wrapper for calculate_document_similarity

Usage

```
semantic_similarity_analysis(...)
```

Arguments

...	Arguments passed to calculate_document_similarity

Value

Similarity analysis results

See Also

`calculate_document_similarity()` for the core computation; `plot_similarity_heatmap()` to render the resulting matrix

sentiment_embedding_analysis

Embedding-based Sentiment Analysis

Description

Performs sentiment analysis using transformer-based embeddings and neural models. This approach uses pre-trained language models for contextual sentiment detection without requiring sentiment lexicons. Particularly effective for handling:

- Complex contextual sentiment
- Implicit sentiment and sarcasm
- Domain-specific sentiment
- Negation and intensifiers (automatically handled by the model)

Usage

```
sentiment_embedding_analysis(
  texts,
  embeddings = NULL,
  model_name = "distilbert-base-uncased-finetuned-sst-2-english",
  doc_names = NULL,
  use_gpu = FALSE
)
```

Arguments

texts	Character vector of texts to analyze
embeddings	Optional pre-computed embedding matrix (from generate_embeddings)
model_name	Sentiment model name (default: "distilbert-base-uncased-finetuned-sst-2-english")
doc_names	Optional document names/IDs
use_gpu	Whether to use GPU if available (default: FALSE)

Value

A list containing:

document_sentiment Data frame with document-level sentiment scores and classifications

emotion_scores NULL (emotion detection not currently supported for embeddings)

summary_stats Summary statistics including document counts and average scores

model_used Name of the transformer model used

feature_type "embeddings"

Examples

```
if (interactive()) {
  abstracts <- TextAnalysisR::SpecialEduTech$abstract[1:10]
  embedding_sentiment <- sentiment_embedding_analysis(abstracts)
  print(embedding_sentiment$document_sentiment)
  print(embedding_sentiment$summary_stats)
}
```

---

sentiment_lexicon_analysis

Analyze Sentiment Using Tidytext Lexicons

Description

Performs lexicon-based sentiment analysis on a DFM object using tidytext lexicons. Supports AFINN, Bing, and NRC lexicons with scoring and emotion analysis.

Tokens absent from the lexicon are ignored (not treated as neutral); `summary_stats$token_match_rate` reports the share of corpus tokens the lexicon covered. AFINN classification uses a +/- 0.5 band on the per-sentiment-word average (`avg_sentiment`); its `n_sentiment_words` column counts matched sentiment tokens, not document length.

Usage

```
sentiment_lexicon_analysis(
  dfm_object,
  lexicon = "bing",
  texts_df = NULL,
  feature_type = "words",
  ngram_range = 2,
  texts = NULL
)
```

Arguments

dfm_object	A quanteda DFM object (unigram)
lexicon	Lexicon to use: "afinn", "bing", or "nrc" (default: "bing")
texts_df	Optional data frame with original texts and metadata (default: NULL)
feature_type	Feature space: "words" (unigrams). The AFINN/Bing/NRC lexicons are unigram lexicons; "ngrams" falls back to unigram scoring with a warning (default: "words").
ngram_range	Retained for backward compatibility; not used for scoring (default: 2)
texts	Optional character vector of texts used to rebuild a unigram DFM (default: NULL)

Value

A list containing:

document_sentiment Data frame with sentiment scores per document

emotion_scores Data frame with emotion scores (NRC only)

summary_stats List of summary statistics

feature_type Feature type used for analysis

Examples

```
abstracts <- TextAnalysisR::SpecialEduTech$abstract[1:10]
corpus <- quanteda::corpus(abstracts)
dfm_object <- quanteda::dfm(quanteda::tokens(corpus))
lexicon_results <- sentiment_lexicon_analysis(dfm_object, lexicon = "bing")
print(lexicon_results$document_sentiment)
```

`setup_python_env` *Setup Python Environment*

Description

Sets up a tiered Python virtual environment.

Usage

```
setup_python_env(envname = "textanalysisr-env", tier = "core", force = FALSE)
```

Arguments

`envname`	Character string name for the virtual environment (default: "textanalysisr-env")
`tier`	Which feature tier to install. One or more of: `"core"` (spacy + pdfplumber, ~200 MB; default), `"embeddings"` (adds sentence-transformers + transformers + torch, ~1 GB), `"topics"` (adds BERTopic + UMAP + HDBSCAN, ~300 MB on top of embeddings).
`force`	Logical, whether to recreate environment if it exists (default: FALSE)

Details

Default `tier = "core"` keeps the install light – spaCy NLP and PDF text extraction only. Add `"embeddings"` for sentence-transformers-based similarity/sentiment, and `"topics"` for BERTopic.

The virtual environment is isolated; system Python is not modified.

Value

Invisible TRUE if successful, stops with error message if failed

Examples

```
if (interactive()) {
  setup_python_env()                                # core only (~200 MB)
  setup_python_env(tier = c("core", "embeddings"))  # +1 GB
  setup_python_env(tier = c("core", "embeddings", "topics"))  # full stack
}
```

spacy_extract_entities *Extract Named Entities with spaCy*

Description

Extract named entities from texts using spaCy NER.

Usage

```
spacy_extract_entities(x, model = "en_core_web_sm")
```

Arguments

x	Character vector of texts OR a quanteda tokens object.
model	Character; spaCy model to use (default: "en_core_web_sm").

Value

A data frame with entity information:

- doc_id: Document identifier
- text: Entity text
- label: Entity type (PERSON, ORG, GPE, etc.)
- start_char: Start character position
- end_char: End character position

spacy_has_vectors *Check if Model Has Word Vectors*

Description

Check if the loaded spaCy model has word vectors for similarity calculations.

Usage

```
spacy_has_vectors()
```

Value

Logical; TRUE if model has vectors, FALSE otherwise.

spacy_initialized *Check if spaCy is Initialized*

Description

Check whether spaCy has been initialized.

Usage

```
spacy_initialized()
```

Value

Logical; TRUE if initialized, FALSE otherwise.

spacy_lemmatize *Lemmatize Texts with spaCy*

Description

Perform lemmatization using spaCy with optimized pipeline settings. Disables unnecessary components (NER, parser) for faster processing.

Usage

```
spacy_lemmatize(x, batch_size = 100, model = "en_core_web_sm")
```

Arguments

x	Character vector of texts OR a quanteda tokens object.
batch_size	Integer; batch size for processing (default: 100).
model	Character; spaCy model to use (default: "en_core_web_sm").

Details

This function disables NER, entity_ruler, and parser components to speed up lemmatization. Use this for lemmas without other annotations.

Value

A data frame with columns: doc_id, token_id, token, lemma.

spacy_parse_full *Parse Texts with spaCy*

Description

Parse texts using spaCy and return token-level annotations. This is the main parsing function for NLP analysis. Works with character vectors or quanteda tokens objects.

Usage

```
spacy_parse_full(
  x,
  pos = TRUE,
  tag = TRUE,
  lemma = TRUE,
  entity = FALSE,
  dependency = FALSE,
  morph = FALSE,
  model = "en_core_web_sm"
)
```

Arguments

x	Character vector of texts OR a quanteda tokens object.
pos	Logical; include coarse POS tags (default: TRUE).
tag	Logical; include fine-grained tags (default: TRUE).
lemma	Logical; include lemmatized forms (default: TRUE).
entity	Logical; include named entity tags (default: FALSE).
dependency	Logical; include dependency relations (default: FALSE).
morph	Logical; include morphological features (default: FALSE).
model	Character; spaCy model to use (default: "en_core_web_sm").

Value

A data frame with token-level annotations including:

- doc_id: Document identifier
- sentence_id: Sentence number within document
- token_id: Token position within sentence
- token: Original token text
- pos: Coarse POS tag (if pos = TRUE)
- tag: Fine-grained tag (if tag = TRUE)
- lemma: Lemmatized form (if lemma = TRUE)

- `entity`: Named entity tag (if entity = TRUE)
- `head_token_id`: Head token ID (if dependency = TRUE)
- `dep_rel`: Dependency relation (if dependency = TRUE)
- `morph`: Morphological features string (if morph = TRUE)

Examples

```
if (interactive()) {
# From SpecialEduTech dataset
texts <- TextAnalysisR::SpecialEduTech$abstract[1:5]
parsed <- spacy_parse_full(texts, morph = TRUE)

# From quanteda tokens
united <- unite_cols(TextAnalysisR::SpecialEduTech, c("title", "abstract"))
tokens <- prep_texts(united, text_field = "united_texts")
parsed <- spacy_parse_full(tokens, morph = TRUE)
}
```

SpecialEduTech *Special education technology bibliographic data*

Description

Contains bibliographic data of journal articles and dissertations on the use of technology in teaching mathematics to students with disabilities.

Usage

```
SpecialEduTech
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 490 rows and 6 columns.

Source

Bibliographic records compiled and coded by the author for Shin, M., Ok, M. W., Choo, S., Hossain, G., Bryant, D. P., & Kang, E. (2023). A content analysis of research on technology use for teaching mathematics to students with disabilities: Word networks and topic modeling. *International Journal of STEM Education*, 10, Article 23. doi:10.1186/s40594023004l4x. Shared as open data at https://github.com/mshin77/math-tech-sped.

Examples

```
SpecialEduTech
```

stm_15 *An example structure of a structural topic model*

Description

Contains 15 topics, topic prevalences, and etc. from `stm::stm`.

Usage

```
stm_15
```

Format

An object of class `STM` of length 11.

Examples

```
stm_15
```

stopwords_list *Stopwords List*

Description

A dataset containing stopwords for text preprocessing

Format

A character vector of stopwords

summarize_morphology *Summarize Morphology Features*

Description

Creates a summary table of morphological feature distributions with counts and percentages for each feature value.

Usage

```
summarize_morphology(data, features = NULL)
```

Arguments

data	Data frame with morph_* columns from extract_morphology().
features	Character vector of features to summarize. If NULL, all available morph_* columns are used.

Value

A data frame with Feature, Value, Count, and Percentage columns.

Examples

```
if (interactive()) {
  tokens <- quanteda::tokens(TextAnalysisR::SpecialEduTech$abstract[1])
  morphology_data <- extract_morphology(tokens)
  summary_table <- summarize_morphology(morphology_data)
  print(summary_table)
}
```

---

unite_cols *Unite Text Columns*

Description

This function unites specified text columns in a data frame into a single column named "united_texts" while retaining the original columns.

Usage

```
unite_cols(df, listed_vars)
```

Arguments

df	A data frame that contains text data.
listed_vars	A character vector of column names to be united into "united_texts".

Value

A data frame with a new column "united_texts" created by uniting the specified variables.

See Also

prep_texts() to tokenize and clean the united text; import_files() to load source data first

Examples

```
mydata <- TextAnalysisR::SpecialEduTech

united_tbl <- TextAnalysisR::unite_cols(
  mydata,
  listed_vars = c("title", "keyword", "abstract")
)
print(united_tbl)
```

`validate_cross_models` *Cross-Analysis Validation*

Description

Performs cross-validation between different analysis types (STM, semantic, clustering).

Usage

```
validate_cross_models(semantic_results, stm_results = NULL, verbose = TRUE)
```

Arguments

`semantic_results`	Results from semantic analysis.
`stm_results`	Optional STM results for comparison.
`verbose`	Logical, if TRUE, prints progress messages.

Value

A list containing cross-validation metrics.

`validate_semantic_coherence`
Validate Semantic Coherence

Description

Validates the semantic coherence of topic assignments using intra-cluster distance in embedding space.

Usage

```
validate_semantic_coherence(embeddings, topic_assignments, ...)
```

Arguments

embeddings	Document embeddings matrix.
topic_assignments	Vector of topic assignments for documents.
...	Additional parameters (currently unused).

Value

List containing coherence score and metrics.

---

word_correlation_network

Analyze and Visualize Word Correlation Networks

Description

This function creates a word correlation network based on a document-feature matrix (dfm).

Usage

```
word_correlation_network(
  dfm_object,
  doc_var = NULL,
  common_term_n = 130,
  corr_n = 0.4,
  top_node_n = 40,
  nrows = 1,
  height = 1000,
  width = 900,
  node_label_size = 22,
  community_method = "leiden",
  node_size_by = "degree",
  node_color_by = "community",
  seed = 123,
  category_params = NULL
)
```

Arguments

dfm_object	A quanteda document-feature matrix (dfm).
doc_var	A document-level metadata variable (default: NULL).
common_term_n	Minimum number of documents a term must appear in (default: 130). This prefilter prevents spurious high correlations from rare words; lower it for small corpora.
corr_n	Minimum phi correlation for keeping an edge (default: 0.4). A heuristic default; report edge counts across nearby thresholds to check sensitivity.

top_node_n	Number of top nodes to display (default: 40).
nrows	Number of rows to display in the table (default: 1).
height	The height of the resulting Plotly plot, in pixels (default: 1000).
width	The width of the resulting Plotly plot, in pixels (default: 900).
node_label_size	Maximum font size for node labels in pixels (default: 22).
community_method	Community detection method: "leiden" (default) or "louvain".
node_size_by	Node sizing method: "degree", "betweenness", "closeness", "eigenvector", or "fixed" (default: "degree").
node_color_by	Node coloring method: "community" or "centrality" (default: "community").
seed	Integer seed for the force-directed layout, so the plot is reproducible (default: 123).
category_params	Optional named list of category-specific parameters. Each element should be a list with common_term_n, corr_n, and top_node_n values for that category (default: NULL).

Value

A list containing the ggplot2 plot, a table, and a summary.

See Also

Other semantic: word_co_occurrence_network()

Examples

```
if (interactive()) {
  df <- TextAnalysisR::SpecialEduTech

  united_tbl <- TextAnalysisR::unite_cols(df, listed_vars = c("title", "abstract"))

  tokens <- TextAnalysisR::prep_texts(united_tbl, text_field = "united_texts")

  dfm_object <- quanteda::dfm(tokens)

  word_correlation_network_results <- TextAnalysisR::word_correlation_network(
                                      dfm_object,
                                      doc_var = NULL,
                                      common_term_n = 30,
                                      corr_n = 0.4,
                                      top_node_n = 40,
                                      nrows = 1,
                                      height = 1000,
                                      width = 900,
                                      community_method = "leiden")
  print(word_correlation_network_results$plot)
```

```
  print(word_correlation_network_results$table)
  print(word_correlation_network_results$summary)
}
```

`word_co_occurrence_network` *Analyze and Visualize Word Co-occurrence Networks*

Description

This function creates a word co-occurrence network based on a document-feature matrix (dfm).

Usage

```
word_co_occurrence_network(
  dfm_object,
  doc_var = NULL,
  co_occur_n = 50,
  edge_metric = c("count", "pmi"),
  top_node_n = 30,
  nrows = 1,
  height = 800,
  width = 900,
  node_label_size = 22,
  community_method = "leiden",
  node_size_by = "degree",
  node_color_by = "community",
  seed = 123,
  category_params = NULL
)
```

Arguments

`dfm_object`	A quanteda document-feature matrix (dfm).
`doc_var`	A document-level metadata variable (default: NULL).
`co_occur_n`	Minimum co-occurrence count (default: 50). A heuristic default; vary it to check edge-count sensitivity.
`edge_metric`	Edge weight: "count" (default, raw co-occurrence) or "pmi" (pointwise mutual information, correcting the bias toward generic high-frequency words; only positive-PMI pairs are kept).
`top_node_n`	Number of top nodes to display (default: 30).
`nrows`	Number of rows to display in the table (default: 1).
`height`	The height of the resulting Plotly plot, in pixels (default: 800).
`width`	The width of the resulting Plotly plot, in pixels (default: 900).

node_label_size
: Maximum font size for node labels in pixels (default: 22).

community_method
: Community detection method: "leiden" (default) or "louvain".

node_size_by
: Node sizing method: "degree", "betweenness", "closeness", "eigenvector", or "fixed" (default: "degree").

node_color_by
: Node coloring method: "community" or "centrality" (default: "community").

seed
: Integer seed for the force-directed layout, so the plot is reproducible (default: 123).

category_params
: Optional named list of category-specific parameters. Each element should be a list with co_occur_n and top_node_n values for that category (default: NULL).

Value

A list containing the ggplot2 plot, a table, and a summary.

See Also

Other semantic: word_correlation_network()

Examples

```
if (interactive()) {
  df <- TextAnalysisR::SpecialEduTech

  united_tbl <- TextAnalysisR::unite_cols(df, listed_vars = c("title", "abstract"))

  tokens <- TextAnalysisR::prep_texts(united_tbl, text_field = "united_texts")

  dfm_object <- quanteda::dfm(tokens)

  word_co_occurrence_network_results <- TextAnalysisR::word_co_occurrence_network(
                                        dfm_object,
                                        doc_var = NULL,
                                        co_occur_n = 50,
                                        top_node_n = 30,
                                        nrows = 1,
                                        height = 800,
                                        width = 900,
                                        community_method = "leiden")
  print(word_co_occurrence_network_results$plot)
  print(word_co_occurrence_network_results$table)
  print(word_co_occurrence_network_results$summary)
}
```

Index