Package 'textpress'

February 23, 2026

Type Package

Title A Lightweight and Versatile NLP Toolkit

Version 1.1.0

Maintainer Jason Timm <JaTimm@salud.unm.edu>

Description A lightweight toolkit for text retrieval and NLP with a consistent and predictable API organized around four actions: fetching, reading, processing, and searching. Functions cover the full pipeline from web data acquisition to text processing and indexing. Multiple search strategies are supported including regex, BM25 keyword ranking, cosine similarity, and dictionary matching. Pipe-friendly with no heavy dependencies and all outputs are plain data frames. Also useful as a building block for retrieval-augmented generation pipelines and autonomous agent workflows.

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 3.5)

Imports data.table, httr, Matrix, rvest, stringi, stringr, xml2, pbapply, jsonlite, lubridate

Suggests SnowballC (>= 0.7.0)

RoxygenNote 7.3.3

URL https://github.com/jaytimm/textpress, https://jaytimm.github.io/textpress/

BugReports https://github.com/jaytimm/textpress/issues

NeedsCompilation no

Author Jason Timm [aut, cre] (year: 2026)

Repository CRAN

Date/Publication 2026-02-23 15:40:02 UTC

Contents

`abbreviations` *Common Abbreviations for Linguistic Processing*

Description

A named list containing common abbreviations used in text analysis.

Usage

```
abbreviations
```

Format

A named list with the following components:

`abbreviations` A character vector of common abbreviations, including titles, months, and standard abbreviations.

Source

Internally compiled linguistic resource.

dict_generations *Demo dictionary of generation-name variants for NER*

Description

A small dictionary of generational cohort terms (Greatest, Silent, Boomers, Gen X, Millennials, Gen Z, Alpha, etc.) and spelling/variant forms, for use with `search_dict`. Built in-package (no `data()`).

Usage

```
dict_generations
```

Format

A data frame with columns `variant` (surface form to match), `TermName` (standardized label), `is_cusp` (logical), `start` and `end` (birth year range; Pew definitions where applicable, see https://github.com/jaytimm/AmericanGenerations/blob/main/data/pew-generations.csv).

Examples

```
head(dict_generations)
# use as term list: search_dict(corpus, by = "doc_id", terms = dict_generations$variant)
```

dict_political *Demo dictionary of political / partisan term variants for NER*

Description

A small dictionary of political party and ideology terms (Democrat, Republican, MAGA, Liberal, Conservative, Christian Nationalist, White Supremacist, etc.) and spelling/variant forms, for use with `search_dict`. Built in-package (no `data()`).

Usage

```
dict_political
```

Format

A data frame with columns `variant` (surface form to match) and `TermName` (standardized label).

Examples

```
head(dict_political)
# search_dict(corpus, by = "doc_id", terms = dict_political$variant)
```

fetch_urls *Fetch URLs from a search engine*

Description

Queries DuckDuckGo Lite and returns result URLs (no local text search). Use `read_urls` to get content from these URLs.

Usage

```
fetch_urls(query, n_pages = 1, date_filter = "w")
```

Arguments

`query`	Search query string.
`n_pages`	Number of DDG Lite pages to fetch (default 1). ~30 results per page.
`date_filter`	Recency filter: `"d"` (day), `"w"` (week), `"m"` (month), or `"none"` (default `"w"`).

Value

A `data.table` with columns `search_engine`, `url`, `is_excluded`.

Examples

```
## Not run:
urls_dt <- fetch_urls("R programming nlp", n_pages = 1)
urls_dt$url

## End(Not run)
```

fetch_wiki_refs *Fetch external citation URLs from Wikipedia*

Description

Searches Wikipedia for a topic, then returns external citation URLs from the first matching page's references section. Use `read_urls` to scrape content from those URLs.

Usage

```
fetch_wiki_refs(query, n = 10)
```

Arguments

`query`	Search phrase (e.g. "January 6 Capitol attack").
`n`	Number of citation URLs to return (default 10).

Value

A character vector of external citation URLs (prefers archived when present).

Examples

```
## Not run:
ref_urls <- fetch_wiki_refs("January 6 Capitol attack", n = 10)
articles <- read_urls(ref_urls)

## End(Not run)
```

`fetch_wiki_urls` *Fetch Wikipedia page URLs by search query*

Description

Uses the MediaWiki API to get Wikipedia article URLs matching a keyword. Does not search your local corpus; it retrieves links from Wikipedia. Use `read_urls` to get article content from these URLs.

Usage

```
fetch_wiki_urls(query, limit = 10)
```

Arguments

`query`	Search phrase (e.g. "117th Congress").
`limit`	Number of page URLs to return (default 10).

Value

A character vector of full Wikipedia article URLs.

Examples

```
## Not run:
wiki_urls <- fetch_wiki_urls("January 6 Capitol attack")
corpus <- read_urls(wiki_urls[1])

## End(Not run)
```

get_search_urls — *Get the search URL(s) used by fetch_urls (for debugging or browser use)*

Description

Page 2+ require POST; only page 1 is a direct browser URL.

Usage

```
get_search_urls(query, n_pages = 1, date_filter = "w")
```

Arguments

query	Search query string.
n_pages	Number of pages (informational for page 2+).
date_filter	Recency filter: "d", "w", "m", or "none" (default "w").

Value

Named character vector of URLs.

nlp_cast_tokens — *Convert Token List to Data Frame*

Description

This function converts a list of tokens into a data frame, extracting and separating document and sentence identifiers if needed.

Usage

```
nlp_cast_tokens(tok)
```

Arguments

tok	A list where each element contains tokens corresponding to a document or a sentence.

Value

A data frame with columns for token name and token.

Examples

```
tok <- list(
  tokens = list(
    "1.1" = c("Hello", "world", "."),
    "1.2" = c("This", "is", "an", "example", "."),
    "2.1" = c("This", "is", "a", "party", "!")
  )
)
dtm <- nlp_cast_tokens(tok)
```

---

nlp_index_tokens *Create a BM25 Search Index*

Description

Create a BM25 Search Index

Usage

```
nlp_index_tokens(tokens, k1 = 1.2, b = 0.75, stem = FALSE)
```

Arguments

tokens	A named list of character vectors.
k1	BM25 saturation parameter.
b	BM25 length normalization.
stem	Logical; if TRUE, stems tokens.

---

nlp_roll_chunks *Roll units into fixed-size chunks with optional context*

Description

Groups consecutive rows at the finest by level (e.g. sentences) into fixed-size chunks and optionally adds surrounding context. Like a rolling window over the leaf units.

Usage

```
nlp_roll_chunks(corpus, by, chunk_size, context_size)
```

Arguments

corpus	A data frame or data.table containing a text column and the identifiers specified in by.
by	A character vector of column names used as unique identifiers. The last column determines the search unit and is the level rolled into chunks (e.g., if by = c("doc_id", "sentence_id"), sentences are rolled into chunks).
chunk_size	Integer. Number of units per chunk.
context_size	Integer. Number of units of context around each chunk.

Value

A data.table with chunk_id, chunk (concatenated text), and chunk_plus_context.

Examples

```
corpus <- data.frame(doc_id = c('1', '1', '2'),
                     sentence_id = c('1', '2', '1'),
                     text = c("Hello world.",
                              "This is an example.",
                              "This is a party!"))
chunks <- nlp_roll_chunks(corpus, by = c('doc_id', 'sentence_id'),
                          chunk_size = 2, context_size = 1)
```

---

nlp_split_paragraphs *Split Text into Paragraphs*

Description

Splits text from the 'text' column of a data frame into individual paragraphs, based on a specified paragraph delimiter.

Usage

```
nlp_split_paragraphs(corpus, paragraph_delim = "\\n+")
```

Arguments

corpus	A data frame or data.table containing a text column and identifier column(s) (e.g. doc_id).
paragraph_delim	A regular expression pattern used to split text into paragraphs.

Value

A data.table with columns: 'doc_id', 'paragraph_id', and 'text'. Each row represents a paragraph, along with its associated document and paragraph identifiers.

Examples

```
corpus <- data.frame(doc_id = c('1', '2'),
                     text = c("Hello world.\n\nMind your business!",
                              "This is an example.n\nThis is a party!"))
paragraphs <- nlp_split_paragraphs(corpus)
```

`nlp_split_sentences` *Split Text into Sentences*

Description

This function splits text from a data frame into individual sentences based on specified columns and handles abbreviations effectively.

Usage

```
nlp_split_sentences(
  corpus,
  by = c("doc_id"),
  abbreviations = textpress::abbreviations
)
```

Arguments

`corpus`	A data frame or data.table containing a `text` column and the identifiers specified in `by`.
`by`	A character vector of column names used as unique identifiers. The last column determines the search unit (e.g., if `by = c("doc_id", "para_id")`, the search returns matches at the paragraph level).
`abbreviations`	A character vector of abbreviations to handle during sentence splitting, defaults to textpress::abbreviations.

Value

A data.table with columns from `by`, plus `sentence_id`, `text`, `start`, `end`.

Examples

```
corpus <- data.frame(doc_id = c('1'),
                     text = c("Hello world. This is an example. No, this is a party!"))
sentences <- nlp_split_sentences(corpus)
```

`nlp_tokenize_text` *Tokenize Text Data (mostly) Non-Destructively*

Description

Tokenizes text from a corpus data frame, preserving structure like capitalization and punctuation.

Usage

```
nlp_tokenize_text(
  corpus,
  by = c("doc_id", "paragraph_id", "sentence_id"),
  include_spans = TRUE,
  method = "word"
)
```

Arguments

`corpus`	A data frame or data.table containing a `text` column and the identifiers specified in `by`.
`by`	A character vector of column names used as unique identifiers. The last column determines the search unit (e.g., if `by = c("doc_id", "para_id")`, the search returns matches at the paragraph level).
`include_spans`	Logical. Include start/end character spans for each token.
`method`	Character. `"word"` or `"biber"`.

Value

A named list of tokens (or list of `tokens` and `spans` if `include_spans = TRUE`).

Examples

```
corpus <- data.frame(doc_id = c('1', '1', '2'),
                     sentence_id = c('1', '2', '1'),
                     text = c("Hello world.",
                              "This is an example.",
                              "This is a party!"))
tokens <- nlp_tokenize_text(corpus, by = c('doc_id', 'sentence_id'))
```

read_urls *Read content from URLs*

Description

Fetches each URL and returns a structured data frame (one row per node: headings, paragraphs, lists). Like `read_csv` or `read_html`: bring an external resource into R. Follows `fetch_urls()` or `fetch_wiki_urls()` in the pipeline: fetch = get locations, read = get text.

Usage

```
read_urls(x, cores = 1, detect_boilerplate = TRUE, remove_boilerplate = TRUE)
```

Arguments

`x`	A character vector of URLs.
`cores`	Number of cores for parallel requests (default 1).
`detect_boilerplate`	Logical. Detect boilerplate (e.g. sign-up, related links).
`remove_boilerplate`	Logical. If `detect_boilerplate` is `TRUE`, remove boilerplate rows; if `FALSE`, keep them and add `is_boilerplate`.

Details

Wikipedia is handled with high-fidelity selectors: `div.mw-parser-output` and `h2`/`h3`/`h4` hierarchy. Use `parent_heading` to see which section each node belongs to. The "External links" section and rows with empty `text` are omitted.

Value

A data frame with `url`, `h1_title`, `date`, `type`, `node_id`, `parent_heading`, `text`, and optionally `is_boilerplate`.

Examples

```
## Not run:
urls <- fetch_urls("R programming", n_pages = 1)$url
nodes <- read_urls(urls[1:3], cores = 1)

## End(Not run)
```

search_dict *Exact n-gram matcher (vector of terms)*

Description

Find a long list of multi-word expressions (MWEs) or terms without regex overhead or partial-match risks. Tokenize corpus, build n-grams, then exact join against `terms`. Word boundaries are respected by design. For categories (e.g. term = "R Project", category = "Software"), left_join your metadata onto the result using `ngram` or `term` as key.

Usage

```
search_dict(corpus, by = c("doc_id"), terms, n_min = 1, n_max = 5)
```

Arguments

`corpus`	The text data (data frame or data.table with `text` and by columns).
`by`	Identifier columns (e.g. `c("doc_id", "sentence_id")`).
`terms`	A character vector of terms/variants to find (e.g. `c("United States", "R Project")`).
`n_min`	Integer. Minimum n-gram size (default 1).
`n_max`	Integer. Maximum n-gram size (default 5).

Value

A data.table with `id`, `start`, `end`, `n`, `ngram`, `term` (the matched term from `terms`).

Examples

```
corpus <- data.frame(doc_id = "1", text = "Gen Z and Millennials use social media.")
search_dict(corpus, by = "doc_id", terms = c("Gen Z", "Millennials", "social media"))
```

search_index *Search the BM25 Index*

Description

Search the BM25 Index

Usage

```
search_index(index, query, n = 10, stem = FALSE)
```

Arguments

index	A data.table created by nlp_index_tokens.
query	A character string.
n	Number of results to return.
stem	Logical; must match the setting used during indexing.

Value

A data.table of results ranked by score.

---

search_regex *Search corpus via regex*

Description

Search corpus via regex

Usage

```
search_regex(corpus, query, by, highlight = c("<b>", "</b>"))
```

Arguments

corpus	A data frame or data.table with a text column.
query	The search pattern (regex).
by	Character vector of identifier columns.
highlight	Length-two character vector (default c('', '')).

---

search_vector *Vector search by cosine similarity*

Description

Returns the top-n matches from an embedding matrix for one or more query vectors. Subject-first: embeddings (haystack) then query (needle), pipe-friendly.

Usage

```
search_vector(embeddings, query, n = 10)
```

Arguments

embeddings	A numeric or sparse matrix of embeddings (rows = searchable units).
query	A character (row name in embeddings), a numeric vector (single query), or a numeric matrix (multiple queries).
n	Number of results to return per query (default 10).

Value

A data frame (or list of data frames if multiple queries are provided) containing the match identifiers and similarity scores.

---

util_fetch_embeddings *Fetch embeddings (Hugging Face utility)*

Description

Fetch embeddings (Hugging Face utility)

Usage

```
util_fetch_embeddings(
  corpus,
  by,
  api_token,
  api_url = "https://router.huggingface.co/hf-inference/models/BAAI/bge-small-en-v1.5"
)
```

Arguments

corpus	A data frame or data.table with a text column.
by	Character vector of column names that identify each text unit.
api_token	Your Hugging Face API token.
api_url	The inference endpoint URL.

Value

A numeric matrix with row names derived from by.

Index