Package 'tall'

February 12, 2026

Title Text Analysis for All

Version 0.5.2

Description An R 'shiny' app designed for diverse text analysis tasks, offering a wide range of methodologies tailored to Natural Language Processing (NLP) needs. It is a versatile, general-purpose tool for analyzing textual data. 'tall' features a comprehensive workflow, including data cleaning, preprocessing, statistical analysis, and visualization, all integrated for effective text analysis.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

URL https://github.com/massimoaria/tall, https://www.k-synth.com/tall/

BugReports https://github.com/massimoaria/tall/issues

Depends R (>= 3.5.0), shiny

Imports base64enc, ca, chromote, curl (>= 6.3.0), doParallel, dplyr (>= 1.1.0), DT, fontawesome, ggplot2, ggraph, ggwordcloud, graphics, httr2, igraph, jsonlite, later, openxlsx, pagedown, parallel, pdftools (>= 3.6.0), plotly, promises, purrr, ranger, Rcpp (>= 1.0.3), readr, readtext, readxl, rlang, RSpectra, shinycssloaders (>= 1.1.0), shinydashboardPlus, shinyFiles, shinyjs, shinyWidgets, sparkline, stringr, strucchange, textrank, tidygraph, tidyr, topicmodels, udpipe, umap, visNetwork, word2vec

LazyData true

LinkingTo Rcpp

NeedsCompilation yes

Author Massimo Aria [aut, cre, cph] (0000-0002-8517-9411), Maria Spano [aut] (ORCID: <https://orcid.org/0000-0002-3103-2342>), Luca D'Aniello [aut] (ORCID: <https://orcid.org/0000-0003-1019-9212>), Corrado Cuccurullo [ctb] (ORCID: <https://orcid.org/0000-0002-7401-8575>), Michelangelo Misuraca [ctb] (ORCID: <https://orcid.org/0000-0002-8794-966X>)

Maintainer Massimo Aria <aria@unina.it>

Repository CRAN

Date/Publication 2026-02-12 08:50:02 UTC

Contents

`calculate_ngram_is` *Calculate IS index for n-grams*

Description

This function calculates the IS (Absorption Index) from Morrone (1996) for all n-grams in the corpus. Only n-grams that start AND end with lexical words are considered.

Usage

```
calculate_ngram_is(
  dfTag,
  max_ngram = 5,
  term = "lemma",
  pos = c("NOUN", "ADJ", "ADV", "VERB"),
  min_freq = 1,
  min_IS_norm = 0
)
```

Arguments

`dfTag`	A data frame with tagged text data containing columns: doc_id, sentence_id, token_id, lemma/token, upos
`max_ngram`	Maximum length of n-grams to generate (default: 5)
`term`	Character string indicating which column to use: "lemma" or "token" (default: "lemma")

pos	Character vector of POS tags considered lexical (default: c("NOUN", "ADJ", "ADV", "VERB"))
min_freq	Minimum frequency threshold for n-grams (default: 1)
min_IS_norm	Minimum normalized IS threshold for n-grams (default: 0)

Details

The IS index is calculated as: IS = (sum 1/freq_i) × freq_ngram × n_lexical where freq_i is the frequency of each word in the n-gram, freq_ngram is the frequency of the n-gram, and n_lexical is the number of lexical words. IS_norm is the normalized version: IS / L^2 where L is the n-gram length.

OPTIMIZATION: Only n-grams that start AND end with lexical words (as defined by the 'pos' parameter) are generated, significantly reducing computation time.

Value

A tibble with columns: ngram, n_length, ngram_freq, n_lexical, IS, IS_norm

Examples

```
## Not run:
IS <- calculate_ngram_is(dfTag, max_ngram = 4, term = "lemma", min_freq = 2)
head(IS)

## End(Not run)
```

---

mobydick *Lemmatized Text of Moby-Dick (Chapters 1-10)*

Description

This dataset contains the lemmatized version of the first 10 chapters of the novel Moby-Dick by Herman Melville. The data is structured as a dataframe with multiple linguistic annotations.

Usage

```
data(mobydick)
```

Format

A dataframe with multiple rows and 26 columns:

doc_id Character: Unique document identifier

paragraph_id Integer: Paragraph index within the document

sentence_id Integer: Sentence index within the paragraph

sentence Character: Original sentence text

start Integer: Start position of the token in the sentence

end Integer: End position of the token in the sentence

term_id Integer: Unique term identifier

token_id Integer: Token index in the sentence

token Character: Original token (word)

lemma Character: Lemmatized form of the token

upos Character: Universal POS tag

xpos Character: Language-specific POS tag

feats Character: Morphological features

head_token_id Integer: Head token in dependency tree

dep_rel Character: Dependency relation label

deps Character: Enhanced dependency relations

misc Character: Additional information

folder Character: Folder containing the document

split_word Character: The word used to separate the chapters in the original book

filename Character: Source file name

doc_selected Logical: Whether the document is selected

POSSelected Logical: Whether POS was selected

sentence_hl Character: Highlighted sentence

docSelected Logical: Whether the document was manually selected

noHapax Logical: Whether hapax legomena were removed

noSingleChar Logical: Whether single-character words were removed

lemma_original_nomultiwords Character: Lemmatized form without multi-word units

Source

Extracted and processed from the text of Moby-Dick by Herman Melville.

Examples

```
data(mobydick)
head(mobydick)
```

process_multiwords_fast *Optimized multiword processing workflow*

Description

Complete optimized workflow for multiword detection and processing. Uses C++ functions and data.table for maximum performance.

Usage

```
process_multiwords_fast(x2, stats, term = c("lemma", "token"))
```

Arguments

x2	Data frame with token information
stats	Data frame with multiword statistics (keyword, ngram columns)
term	Type of term to process: "lemma" or "token"

Details

This function replaces the original switch block with an optimized version that uses:

- C++ functions for text recoding
- Vectorized operations instead of multiple mutate calls
- Pre-computed lookups to avoid repeated joins

Value

Data frame with columns: doc_id, term_id, multiword, upos_multiword, ngram

Examples

```
## Not run:
result <- process_multiwords_fast(dfTag, multiword_stats, term = "lemma")

## End(Not run)
```

`reinert` *Segment clustering based on the Reinert method - Simple clustering*

Description

Segment clustering based on the Reinert method - Simple clustering

Usage

```
reinert(
  x,
  k = 10,
  term = "token",
  segment_size = 40,
  min_segment_size = 3,
  min_split_members = 5,
  cc_test = 0.3,
  tsj = 3
)
```

Arguments

`x`	tall data frame of documents
`k`	maximum number of clusters to compute
`term`	indicates the type of form "lemma" or "token". Default value is term = "lemma".
`segment_size`	number of forms by document. Default value is segment_size = 40
`min_segment_size`	minimum number of forms by document. Default value is min_segment_size = 5
`min_split_members`	minimum number of segment in a cluster
`cc_test`	contingency coefficient value for feature selection
`tsj`	minimum frequency value for feature selection

Details

See the references for original articles on the method. Special thanks to the authors of the rainette package (https://github.com/juba/rainette) for inspiring the coding approach used in this function.

Value

The result is a list of both class `hclust` and `reinert_tall`.

References

- Reinert M, Une methode de classification descendante hierarchique: application à l'analyse lexicale par contexte, Cahiers de l'analyse des donnees, Volume 8, Numéro 2, 1983. https://www.numdam.org/item/?id=CAD_1983__8_2_187_0
- Reinert M., Alceste une méthodologie d'analyse des données textuelles et une application: Aurelia De Gerard De Nerval, Bulletin de Methodologie Sociologique, Volume 26, Numero 1, 1990. doi:10.1177/075910639002600103
- Barnier J., Privé F., rainette: The Reinert Method for Textual Data Clustering, 2023, doi:10.32614/CRAN.package.rainette

Examples

```
data(mobydick)
res <- reinert(
  x = mobydick,
  k = 10,
  term = "token",
  segment_size = 40,
  min_segment_size = 5,
  min_split_members = 10,
  cc_test = 0.3,
  tsj = 3
)
```

---

reinPlot *Plot Terms by Cluster*

Description

This function creates a horizontal bar plot to visualize the most significant terms for each cluster, based on their Chi-squared statistics.

Usage

```
reinPlot(terms, nPlot = 10)
```

Arguments

`terms`	A data frame containing terms and their associated statistics, such as Chi-squared values, generated by the `term_per_cluster` function. The data frame must include the following columns: • `term`: The term to plot. • `chi_square`: The Chi-squared statistic associated with the term. • `sign`: The sign of the term (`"positive"` or `"negative"`).
`nPlot`	Integer. The number of top terms to plot for each sign (`"positive"` and `"negative"`). Default is 10.

Details

The function organizes the input data by Chi-squared values and selects the top terms for each sign. The plot uses different colors for positive and negative terms, with hover tooltips providing detailed information.

Value

An interactive horizontal bar plot (using `plotly`) displaying the top terms for each cluster. The plot includes:

- Bars representing the Chi-squared values of terms.
- Hover information displaying the term and its Chi-squared value.

See Also

`term_per_cluster`

Examples

```
## Not run:
data(mobydick)
res <- reinert(
  x = mobydick,
  k = 10,
  term = "token",
  segment_size = 40,
  min_segment_size = 5,
  min_split_members = 10,
  cc_test = 0.3,
  tsj = 3
)

tc <- term_per_cluster(res, cutree = NULL, k = 1, negative = FALSE)

fig <- reinPlot(tc$terms, nPlot = 10)

## End(Not run)
```

`reinSummary` *Summarize Reinert Clustering Results*

Description

This function summarizes the results of the Reinert clustering algorithm, including the most frequent documents and significant terms for each cluster. The input is the result returned by the `term_per_cluster` function.

Usage

```
reinSummary(tc, n = 10)
```

Arguments

tc	A list returned by the term_per_cluster function. The list includes: • segments: A data frame with segments information, including cluster and doc_id. • terms: A data frame with terms information, including cluster, sign, chi_square, and term.
n	Integer. The number of top terms (based on Chi-squared value) to include in the summary for each cluster and sign. Default is 10.

Details

This function performs the following steps:

1. Extracts the most frequent document for each cluster.
2. Summarizes the number of documents per cluster.
3. Selects the top n terms for each cluster, separated by positive and negative signs.
4. Combines the terms and segment information into a final summary table.

Value

A data frame summarizing the clustering results. The table includes:

- cluster: The cluster ID.
- Positive terms: The top n positive terms for each cluster, concatenated into a single string.
- Negative terms: The top n negative terms for each cluster, concatenated into a single string.
- Most frequent document: The document ID that appears most frequently in each cluster.
- N. of Documents per Cluster: The number of documents in each cluster.

See Also

term_per_cluster, reinPlot

Examples

```
data(mobydick)
res <- reinert(
  x = mobydick,
  k = 10,
  term = "token",
  segment_size = 40,
  min_segment_size = 5,
  min_split_members = 10,
  cc_test = 0.3,
  tsj = 3
```

```
)

tc <- term_per_cluster(res, cutree = NULL, k = 1:10, negative = FALSE)

S <- reinSummary(tc, n = 10)

head(S, 10)
```

---

tall *TALL UI*

Description

tall performs text analysis for all.

Usage

```
tall(
  host = "127.0.0.1",
  port = NULL,
  launch.browser = TRUE,
  maxUploadSize = 1000
)
```

Arguments

host	The IPv4 address that the application should listen on. Defaults to the shiny.host option, if set, or "127.0.0.1" if not.
port	is the TCP port that the application should listen on. If the port is not specified, and the shiny.port option is set (with options(shiny.port = XX)), then that port will be used. Otherwise, use a random port.
launch.browser	If true, the system's default web browser will be launched automatically after the app is started. Defaults to true in interactive sessions only. This value of this parameter can also be a function to call with the application's URL.
maxUploadSize	is a integer. The max upload file size argument. Default value is 1000 (megabyte)

Value

No return value, called for side effects.

`term_per_cluster` *Extract Terms and Segments for Document Clusters*

Description

This function processes the results of a document clustering algorithm based on the Reinert method. It computes the terms and their significance for each cluster, as well as the associated document segments.

Usage

```
term_per_cluster(res, cutree = NULL, k = 1, negative = TRUE)
```

Arguments

`res`	A list containing the results of the Reinert clustering algorithm. Must include at least `dtm` (a document-term matrix) and `corresp_uce_uc_full` (a correspondence between segments and clusters).
`cutree`	A custom cutree structure. If `NULL`, the default `cutree_reinart` is used to determine cluster membership.
`k`	A vector of integers specifying the clusters to analyze. Default is `1`.
`negative`	Logical. If `TRUE`, include negative terms in the results. If `FALSE`, exclude them. Default is `TRUE`.

Details

The function integrates document-term matrix rows for missing segments, calculates term statistics for each cluster, and filters terms based on their significance. Terms can be excluded based on their significance (`signExcluded`).

Value

A list with the following components:

`terms` A data frame of significant terms for each cluster. Columns include:

- `chi_square`: Chi-squared statistic for the term.
- `p_value`: P-value of the chi-squared test.
- `sign`: Significance of the term (`positive`, `negative`, or `none`).
- `term`: The term itself.
- `freq`: Observed frequency of the term in the cluster.
- `indep`: Expected frequency of the term under independence.
- `cluster`: The cluster ID.

`segments` A data frame of document segments associated with each cluster. Columns include:

- `uc`: Unique segment identifier.

- doc_id: Document ID for the segment.
- cluster: Cluster ID.
- segment: The text content of each segment.

Examples

```
data(mobydick)
res <- reinert(
  x = mobydick,
  k = 10,
  term = "token",
  segment_size = 40,
  min_segment_size = 5,
  min_split_members = 10,
  cc_test = 0.3,
  tsj = 3
)

tc <- term_per_cluster(res, cutree = NULL, k = 1:10, negative = FALSE)

head(tc$segments, 10)

head(tc$terms, 10)
```

---

txt_recode_fast *Fast text recoding (Rcpp version)*

Description

Efficiently recodes text values using C++ hash tables. This is a drop-in replacement for txt_recode but significantly faster for large vectors.

Usage

```
txt_recode_fast(x, from = c(), to = c(), na.rm = FALSE)
```

Arguments

x	A character vector to recode
from	A character vector with values of x which you want to recode
to	A character vector with values you want to use to recode to
na.rm	Logical, if set to TRUE, will put all values of x which have no matching value in from to NA. Defaults to FALSE

Details

This function uses C++ hash tables for O(1) lookup time, making it much faster than the pure R implementation, especially for large datasets.

Performance improvement: ~50-100x faster than base R `txt_recode` for vectors with 100K+ elements.

Value

A character vector of the same length as `x` where values matching `from` are replaced by corresponding values in `to`

Examples

```
x <- c("NOUN", "VERB", "NOUN", "ADV")
txt_recode_fast(x,
  from = c("VERB", "ADV"),
  to = c("conjugated verb", "adverb")
)
```

`txt_recode_ngram_fast` *Fast n-gram recoding for multiword detection*

Description

Efficiently combines consecutive tokens into multiword expressions using C++. This function scans text sequentially to identify and merge n-gram patterns.

Usage

```
txt_recode_ngram_fast(x, compound, ngram, sep = " ")
```

Arguments

`x`	Character vector of tokens (e.g., lemmas or tokens)
`compound`	Character vector of multiword expressions to match
`ngram`	Integer vector indicating the length of each compound
`sep`	String separator to use when joining tokens (default: " ")

Details

When a multiword match is found:

- The first position gets the combined multiword expression
- Subsequent positions that were merged are set to NA

The function checks n-grams from longest to shortest to prioritize longer matches.

Performance: ~80-150x faster than pure R implementation for typical text data.

Value

Character vector where matched n-grams are combined and subsequent tokens (that were merged) are set to NA

Examples

```
tokens <- c("machine", "learning", "is", "cool", "machine", "learning")
compounds <- c("machine learning")
ngrams <- c(2)
txt_recode_ngram_fast(tokens, compounds, ngrams, " ")
# Returns: c("machine learning", NA, "is", "cool", "machine learning", NA)
```

Index