Package ‘gibasa’

June 3, 2023

Type Package
Title An Alternative ‘Rcpp’ Wrapper of ‘MeCab’
Version 0.9.4
Maintainer Akiru Kato <paithiov909@gmail.com>
Description A plain ‘Rcpp’ wrapper of ‘MeCab’ that can segment Chinese, Japanese, and Korean text into tokens. The main goal of this package is to provide an alternative to ‘tidytext’ using morphological analysis.
License GPL (>= 3)
URL https://paithiov909.github.io/gibasa/
BugReports https://github.com/paithiov909/gibasa/issues
Imports audubon (>= 0.5.0), dplyr, purrr, Rcpp, RcppParallel, rlang (>= 0.4.11), stringi, utils
Suggests roxygen2, testthat (>= 3.0.0)
LinkingTo Rcpp, RcppParallel
Config/Needs/website ggdendro, ggpubr, quanteda, quanteda.textmodels, quanteda.textplots, quanteda.textstats, reactable, stringr, textmineR, textrecipes, tidymodels, topicmodels, uwot, xgboost, paithiov909/ldccr
Config/testthat/edition 3
Encoding UTF-8
RoxygenNote 7.2.3
SystemRequirements GNU make, MeCab
NeedsCompilation yes
Author Akiru Kato [aut, cre], Shogo Ichinose [aut], Taku Kudo [aut], Jorge Nocedal [ctb], Nippon Telegraph and Telephone Corporation [cph]
Repository CRAN
Date/Publication 2023-06-03 09:50:02 UTC
as_tokens 

Create a list of tokens

Description

Create a list of tokens

Usage

as_tokens(
  tbl,
  token_field = "token",
  pos_field = get_dict_features()[1],
  nm = NULL
)

Arguments

tbl 
A tibble of tokens out of tokenize().
token_field 
Column name that contains tokens.
pos_field 
Feature name that will be kept as the names of tokens. If you don’t need them, give a NULL for this argument.

nm 
Names of returned list. If left with NULL, "doc_id" field of tbl is used instead.

Value

A named list of tokens.
### Description

Calculates and binds the importance of bigrams and their synergistic average.

### Usage

```r
bind_lr(tbl, term = "token", lr_mode = c("n", "dn"), avg_rate = 1)
```

### Arguments

- **tbl**: A tidy text dataset.
- **term**: Column containing terms as string or symbol.
- **lr_mode**: Method for computing 'FL' and 'FR' values. 'n' is equivalent to 'LN' and 'RN', and 'dn' is equivalent to 'LDN' and 'RDN'.
- **avg_rate**: Weight of the 'LR' value.

### Details

The 'LR' value is the synergistic average of bigram importance that based on the words and their positions (left or right side).

### Value

A data.frame.

### See Also

[doi:10.5715/jnlp.10.27](https://doi.org/10.5715/jnlp.10.27)
Examples

```r
## Not run:
df <- tokenize(
  data.frame(
    doc_id = seq_along(audubon::polano[5:8]),
    text = audubon::polano[5:8]
  )
)
bind_lr(df)

## End(Not run)
```

### bind_tf_idf2

Bind the term frequency and inverse document frequency

Description

Calculates and binds the term frequency, inverse document frequency, and TF-IDF of the dataset. This function experimentally supports 3 types of term frequencies and 4 types of inverse document frequencies, which are implemented in `RMeCab` package.

Usage

```r
bind_tf_idf2(
  tbl,
  term = "token",
  document = "doc_id",
  n = "n",
  tf = c("tf", "tf2", "tf3"),
  idf = c("idf", "idf2", "idf3", "idf4"),
  norm = FALSE,
  rmecab_compat = TRUE
)
```

Arguments

- **tbl**: A tidy text dataset.
- **term**: Column containing terms as string or symbol.
- **document**: Column containing document IDs as string or symbol.
- **n**: Column containing document-term counts as string or symbol.
- **tf**: Method for computing term frequency.
- **idf**: Method for computing inverse document frequency.
- **norm**: Logical; If passed as TRUE, the raw term counts are normalized being divided with L2 norms before computing IDF values.
- **rmecab_compat**: Logical; If passed as TRUE, computes values while taking care of compatibility with `RMeCab`. Note that `RMeCab` always computes IDF values using term frequency rather than raw term counts, and thus TF-IDF values may be doubly affected by term frequency.
**Details**

Types of term frequency can be switched with `tf` argument:

- `tf` is term frequency (not raw count of terms).
- `tf2` is logarithmic term frequency of which base is 10.
- `tf3` is binary-weighted term frequency.

Types of inverse document frequencies can be switched with `idf` argument:

- `idf` is inverse document frequency of which base is 2, with smoothed. 'smoothed' here means just adding 1 to raw counts after logarithmizing.
- `idf2` is global frequency IDF.
- `idf3` is probabilistic IDF of which base is 2.
- `idf4` is global entropy, not IDF in actual.

**Value**

A data.frame.

**Examples**

```r
## Not run:
df <- tokenize(
data.frame(
  doc_id = seq_along(audubon::polano[5:8]),
  text = audubon::polano[5:8]
)
) |> dplyr::group_by(doc_id) |>
dplyr::count(token) |>
dplyr::ungroup()
bind_tf_idf2(df)
## End(Not run)
```

---

**collapse_tokens**

Collapse sequences of tokens by condition

**Description**

Concatenates sequences of tokens in the tidy text dataset, while grouping them by an expression.

**Usage**

```r
collapse_tokens(tbl, condition, .collapse = "")
```
Arguments

- **tbl**: A tidy text dataset.
- **condition**: A logical expression.
- **.collapse**: String with which tokens are concatenated.

Details

Note that this function drops all columns except but 'token' and columns for grouping sequences. So, the returned data.frame has only 'doc_id', 'sentence_id', 'token_id', and 'token' columns.

Value

A data.frame.

Examples

```r
## Not run:
df <- tokenize(
data.frame(
  doc_id = seq_along(audubon::polano[5:8]),
  text = audubon::polano[5:8]
)
) |> prettify(col_select = "POS1")

head(collapse_tokens(
  df,
  POS1 == "\u540d\u8a5e" & stringr::str_detect(token, "^[\p{Han}]\+$")
))
## End(Not run)
```

---

**dictionary_info**  
Get dictionary information

Description

Get dictionary information

Arguments

- **sys_dic**: String scalar.
- **user_dic**: String scalar.

Value

data.frame.
gbs_tokenize

Tokenize sentences using 'MeCab'

Description

Tokenize sentences using 'MeCab'

Usage

```r
gbs_tokenize(
  x,
  sys_dic = "",  # Character scalar; path to the system dictionary for mecab. Note that the system dictionary is expected to be compiled with UTF-8, not Shift-JIS or other encodings.
  user_dic = "",  # Character scalar; path to the user dictionary for mecab.
  split = FALSE,  # Logical. When passed as TRUE, the function internally splits the sentences into sub-sentences using stringi::stri_split_boundaries(type = "sentence").
  partial = FALSE,  # Logical. When passed as TRUE, activates partial parsing mode. To activate this feature, remember that all spaces at the start and end of the input chunks are already squashed. In particular, trailing spaces of chunks sometimes cause fatal errors.
  mode = c("parse", "wakati")
)
```

Arguments

- `x`: A data.frame like object or a character vector to be tokenized.
- `sys_dic`: Character scalar; path to the system dictionary for mecab.
- `user_dic`: Character scalar; path to the user dictionary for mecab.
- `split`: Logical. When passed as TRUE, the function internally splits the sentences into sub-sentences using `stringi::stri_split_boundaries(type = "sentence")`.
- `partial`: Logical. When passed as TRUE, activates partial parsing mode. To activate this feature, remember that all spaces at the start and end of the input chunks are already squashed. In particular, trailing spaces of chunks sometimes cause fatal errors.
- `mode`: Character scalar to switch output format.

Value

A tibble or a named list of tokens.
get_dict_features  Get dictionary’s features

Description

Returns dictionary’s features. Currently supports "unidic17" (2.1.2 src schema), "unidic26" (2.1.2 bin schema), "unidic29" (schema used in 2.2.0, 2.3.0), "cc-cedict", "ko-dic" (mecab-ko-dic), "naist11", "sudachi", and "ipa".

Usage

get_dict_features(
  dict = c("ipa", "unidic17", "unidic26", "unidic29", "cc-cedict", "ko-dic", "naist11", "sudachi")
)

Arguments

dict  Character scalar; one of "ipa", "unidic17", "unidic26", "unidic29", "cc-cedict", "ko-dic", "naist11", or "sudachi".

Value

A character vector.

See Also

See also 'CC-CEDICT-MeCab', and 'mecab-ko-dic'.

is_blank  Check if scalars are blank

Description

Check if scalars are blank

Usage

is_blank(x, trim = TRUE, ...)

Arguments

x  Object to check its emptiness.
trim  Logical.
...  Additional arguments for base::sapply().
**lex_density**

Value

Logical.

Examples

```r
is_blank(list(c(a = "", b = NA_character_), NULL))
```

---

**Calculate lexical density**

**Description**

The lexical density is the proportion of content words (lexical items) in documents. This function is a simple helper for calculating the lexical density of given datasets.

**Usage**

```r
lex_density(vec, contents_words, targets = NULL, negate = c(FALSE, FALSE))
```

**Arguments**

- **vec**: A character vector.
- **contents_words**: A character vector containing values to be counted as contents words.
- **targets**: A character vector with which the denominator of lexical density is filtered before computing values.
- **negate**: A logical vector of which length is 2. If passed as TRUE, then respectively negates the predicate functions for counting contents words or targets.

**Value**

A numeric vector.

**Examples**

```r
## Not run:
df <- tokenize(
data.frame(
  doc_id = seq_along(audubon::polano[5:8]),
  text = audubon::polano[5:8]
)
)
df |> prettify(col_select = "POS1") |> dplyr::group_by(doc_id) |> dplyr::summarise(
noun_ratio = lex_density(POS1, "\u540d\u8a5e", "\u5a9e\u52a9\u8a5e", "\u52a9\u52d5\u8a5e", "\u540d\u52a9\u52d5\u8a5e"),
```
mute_tokens

Mute tokens by condition

Description

Permutes tokens in the tidy text dataset with a string scalar only if they are matched to an expression.

Usage

mute_tokens(tbl, condition, .as = NA_character_)

Arguments

- `tbl`: A tidy text dataset.
- `condition`: A logical expression.
- `.as`: String with which tokens are replaced when they are matched to condition. The default value is `NA_character_`.

Value

A data.frame.

Examples

```r
## Not run:
df <- tokenize(
data.frame(
  doc_id = seq_along(audubon::polano[5:8]),
  text = audubon::polano[5:8]
)
)
head(mute_tokens(df, POS1 %in% c(\u5f62\u5bb9\u8a5e", \"\u52a9\u8a5e", \"\u9023\u4f53\u8a5e\"),
    \"\u52d5\u8a5e\"
  )
)
## End(Not run)
```
pack

Pack prettified data.frame of tokens

Description

Packs a data.frame of tokens into a new data.frame of corpus, which is compatible with the Text Interchange Formats.

Usage

pack(tbl, pull = "token", n = 1L, sep = "-", .collapse = " ")

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tbl</td>
<td>A data.frame of tokens.</td>
</tr>
<tr>
<td>pull</td>
<td>Column to be packed into text or ngrams body. Default value is token.</td>
</tr>
<tr>
<td>n</td>
<td>Integer internally passed to ngrams tokenizer function created of audubon::ngram_tokenizer()</td>
</tr>
<tr>
<td>sep</td>
<td>Character scalar internally used as the concatenator of ngrams.</td>
</tr>
<tr>
<td>.collapse</td>
<td>This argument is passed to stringi::stri_join().</td>
</tr>
</tbody>
</table>

Value

A tibble.

Text Interchange Formats (TIF)

The Text Interchange Formats (TIF) is a set of standards that allows R text analysis packages to target defined inputs and outputs for corpora, tokens, and document-term matrices.

Valid data.frame of tokens

The data.frame of tokens here is a data.frame object compatible with the TIF.

A TIF valid data.frame of tokens are expected to have one unique key column (named doc_id) of each text and several feature columns of each tokens. The feature columns must contain at least token itself.

See Also

https://github.com/ropenscilabs/tif
Examples

```r
## Not run:
df <- tokenize(
data.frame(
  doc_id = seq_along(audubon::polano[5:8]),
  text = audubon::polano[5:8]
)
)
pack(df)

## End(Not run)
```

---

**prettify**

_Prettify tokenized output_

Description

Turns a single character column into features separating with delimiter.

Usage

```r
prettify(tbl, into = get_dict_features("ipa"), col_select = seq_along(into))
```

Arguments

- `tbl`: A data.frame that has feature column to be prettified.
- `into`: Character vector that is used as column names of features.
- `col_select`: Character or integer vector that will be kept in prettified features.

Value

A data.frame.

Examples

```r
## Not run:
df <- tokenize(
data.frame(
  doc_id = seq_along(audubon::polano[5:8]),
  text = audubon::polano[5:8]
)
)
pack(df)

prettify(df, col_select = 1:3)
prettify(df, col_select = c(1, 3, 6))
prettify(df, col_select = c("POS1", "Original"))

## End(Not run)
```
Description

Tokenize sentences using 'MeCab'

Usage

tokenize(
  x,
  text_field = "text",
  docid_field = "doc_id",
  sys_dic = "",
  user_dic = "",
  split = FALSE,
  partial = FALSE,
  grain_size = 1L,
  mode = c("parse", "wakati")
)

Arguments

x
<data-masked> A data.frame like object or a character vector to be tokenized.
text_field
<data-masked> String or symbol; column name where to get texts to be tokenized.

docid_field
<data-masked> String or symbol; column name where to get identifiers of texts.
sys_dic
Character scalar; path to the system dictionary for mecab. Note that the system dictionary is expected to be compiled with UTF-8, not Shift-JIS or other encodings.
user_dic
Character scalar; path to the user dictionary for mecab.
split
Logical. When passed as TRUE, the function internally splits the sentences into sub-sentences using stringi::stri_split_boundaries(type = "sentence").
partial
Logical. When passed as TRUE, activates partial parsing mode. To activate this feature, remember that all spaces at the start and end of the input chunks are already squashed. In particular, trailing spaces of chunks sometimes cause fatal errors.
grain_size
Integer value larger than 1. This argument is internally passed to RcppParallel::parallelFor function. Setting a larger chunk size could improve the performance in some cases.
mode
Character scalar to switch output format.

Value

A tibble or a named list of tokens.
## Examples

```r
## Not run:
df <- tokenize(
data.frame(
  doc_id = seq_along(audubon::polano[5:8]),
  text = audubon::polano[5:8]
)
)
head(df)

## End(Not run)
```
Index

as_token, 2
bind_lr, 3
bind_tf_idf2, 4
collapse_tokens, 5
dictionary_info, 6
gbs_tokenize, 7
get_dict_features, 8
is_blank, 8
lex_density, 9
mute_tokens, 10
pack, 11
prettify, 12
tokenize, 13