Package ‘audubon’

May 2, 2023

Title Japanese Text Processing Tools
Version 0.5.1
Description A collection of Japanese text processing tools for filling
Japanese iteration marks, Japanese character type conversions,
segmentation by phrase, and text normalization which is based on rules
for the ‘Sudachi’ morphological analyzer and the ‘NEologd’ (Neologism
dictionary for ‘MeCab’). These features are specific to Japanese and
are not implemented in ‘ICU’ (International Components for Unicode).
License Apache License (>= 2)
URL https://github.com/paithiov909/audubon,
       https://paithiov909.github.io/audubon/
BugReports https://github.com/paithiov909/audubon/issues
Depends R (>= 2.10)
Imports dplyr (>= 1.1.0), magrittr, Matrix, memoise, purrr, readr,
       rlang (>= 0.4.11), stringi, V8
Suggests roxygen2, spelling, testthat (>= 3.0.0)
Config/testthat/edition 3
Encoding UTF-8
Language en-US
LazyData true
RoxygenNote 7.2.3
NeedsCompilation no
Author Akiru Kato [cre, aut],
       Koki Takahashi [cph] (Author of japanese.js),
       Shuhei Iitsuka [cph] (Author of budoux),
       Taku Kudo [cph] (Author of TinySegmenter)
Maintainer Akiru Kato <paithiov909@gmail.com>
Repository CRAN
Date/Publication 2023-05-02 19:00:10 UTC
bind_lr

### 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

Examples

```r
## Not run:
prettify(hiroba, col_select = "POS1") |>
mute_tokens(POS1 != "\u540d\u8a5e") |>
bind_lr()
## End(Not run)
```

bind_tf_idf2

**Bind 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
)
```
bind_tf_idf2

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 by dividing 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 doubled 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 <- dplyr::add_count(hiroba, doc_id, token)
bind_tf_idf2(df)
## End(Not run)
```
collapse_tokens

Description

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

Usage

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 <- prettify(head(hiroba), col_select = "POS1")
collapse_tokens(df, POS1 == "\u540d\u8a5e")
```

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

```r
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'.

Examples

```r
get_dict_features("ipa")
```

---

| hiroba | "Whole tokens of 'Porano no Hiroba' written by Miyazawa Kenji from Aozora Bunko" |

Description

A tidy text data of `audubon::polano` that tokenized with `MeCab`.

Usage

```r
hiroba
```

Format

An object of class `data.frame` with 26849 rows and 5 columns.

Examples

```r
head(hiroba)
```
**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
head(hiroba) |> 
  prettify(col_select = "POS1") |> 
  dplyr::group_by(doc_id) |> 
  dplyr::summarise(
    noun_ratio = lex_density(POS1, 
      "\u540d\u8a5e", 
      c("\u52a9\u8a5e", "\u52a9\u52d5\u8a5e"), 
      negate = c(FALSE, TRUE)
    ),
    mvr = lex_density( 
      POS1, 
      c("\u5f62\u5bb9\u8a5e", "\u526f\u8a5e", "\u9023\u4f53\u8a5e"), 
      "\u52d5\u8a5e"
    ),
    vnr = lex_density(POS1, "\u52d5\u8a5e", "\u540d\u8a5e")
  )
```
mute_tokens  
*Mute tokens by condition*

**Description**

Permutates 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**

```
df <- prettify(head(hiroba), col_select = "POS1")
mute_tokens(df, POS1 %in% c("\u52a9\u8a5e", "\u52a9\u52d5\u8a5e"))```

---

**ngram_tokenizer**

*Ngrams tokenizer*

**Description**

Make an ngram tokenizer function.

**Usage**

```
ngram_tokenizer(n = 1L)```

**Arguments**

- `n`: Integer.

**Value**

ngram tokenizer function
Pack a 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

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

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

pack(strj_tokenize(polano[1:5], format = "data.frame"))
polano

Whole text of ‘Porano no Hiroba’ written by Miyazawa Kenji from Aozora Bunko

Description

Whole text of ‘Porano no Hiroba’ written by Miyazawa Kenji from Aozora Bunko

Usage

polano

Format

An object of class character of length 899.

Details

A dataset containing the text of Miyazawa Kenji’s novel "Porano no Hiroba" which was published in 1934, the year after Kenji’s death. Copyright of this work has expired since more than 70 years have passed after the author’s death.

The UTF-8 plain text is sourced from https://www.aozora.gr.jp/cards/000081/card1935.html and is cleaned of meta data.

Source


Examples

head(polano)

prettify

Prettify tokenized output

Description

Turns a single character column into features separating with delimiter.

Usage

prettify(  
tbl,  
col = "feature",  
into = get_dict_features("ipa"),  
col_select = seq_along(into),  
delim = ","  
)
Arguments

tbl A data.frame that has feature column to be prettified.
col Column name where 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.
delim Character scalar used to separate fields within a feature.

Value

A data.frame.

Examples

prettify(
data.frame(x = c("x,y", "y,z", "z,x")),
col = "x",
into = c("a", "b"),
col_select = "b"
)

Description

Read a rewrite.def file

Usage

read_rewrite_def(
  def_path = system.file("def/rewrite.def", package = "audubon")
)

Arguments

def_path Character scalar; path to the rewriting definition file.

Value

A list.

Examples

str(read_rewrite_def())
strj_fill_iter_mark  Fill Japanese iteration marks

Description
Fills Japanese iteration marks (Odori-ji) with their previous characters if the element has more than 5 characters.

Usage
strj_fill_iter_mark(text)

Arguments
text  Character vector.

Value
A character vector.

Examples
strj_fill_iter_mark(c("\u3042\u3044\u3046\u309d\u3003\u304b\u304d",
"\u91d1\u5b50\u307f\u3059\u309e",
"\u306e\u305f\u308a\u3033\u3035\u304b\u306a",
"\u3057\u308d\uff0f\uff3c\u3068\u3057\u305f")
))

strj_hiraganize  Hiraganize Japanese characters

Description
Converts Japanese katakana to hiragana. It is almost similar to stringi::stri_trans_general(text, "kana-hira"), however, this implementation can also handle some additional symbols such as Japanese kana ligature (aka. goryaku-gana).

Usage
strj_hiraganize(text)

Arguments
text  Character vector.
Value
A character vector.

Examples

```r
strj_hiraganize(
  c(
    paste0(
      "\u3042\u306e\u30a4\u30fc\u30cf\u30c8",
      "\u30fc\u30f4\u30a9\u306e\u3059\u304d",
      "\u3068\u304a\u3063\u305f\u98a8"
    ),
    "\u677f\u57a3\u6b7b\u30b9\u0002a708"
  )
)
```

---

**strj_katakanaize**

**Katakana Japanese characters**

Description

Converts Japanese hiragana to katakana. It is almost similar to `stringi::stri_trans_general(text, "hira-kana")`, however, this implementation can also handle some additional symbols such as Japanese kana ligature (aka. goryaku-gana).

Usage

```r
strj_katakanaize(text)
```

Arguments

- **text**  
  Character vector.

Value

A character vector.

Examples

```r
strj_katakanaize(
  c(
    paste0(
      "\u3042\u306e\u30a4\u30fc\u30cf\u30c8",
      "\u30fc\u30f4\u30a9\u306e\u3059\u304d",
      "\u3068\u304a\u3063\u305f\u98a8"
    ),
    "\u677f\u57a3\u6b7b\u30b9\u0002a708"
  )
)
```
### strj_normalize

**Convert text following the rules of 'NEologd'**

**Description**

Converts characters into normalized style following the rule that is recommended by the Neologism dictionary for 'MeCab'.

**Usage**

```r
strj_normalize(text)
```

**Arguments**

- `text`: Character vector to be normalized.

**Value**

A character vector.

**See Also**


**Examples**

```r
strj_normalize(
  paste0(
    \"\u2015\u2015\u5357\u30a2\u30eb\u30d7\u30b9", 
    \"\u306e\u5929\u7136\u6c34~\u3000\uff33", 
    \uff50\uff41\uff52\uff4f\uff49\uff4e\uff47*", 
    \"\uff40\uff45\uff4f\uff4d\uff4f\uff4e+", 
    \"\u3000\u30ec\u30e2\u30f3\u4e00\u7d5e\u308a"
  )
)
```

### strj_rewrite_as_def

**Rewrite text using rewrite.def**

**Description**

Rewrites text using a 'rewrite.def' file.

**Usage**

```r
strj_rewrite_as_def(text, as = read_rewrite_def())
```
**strj_romanize**

**Romanize Japanese Hiragana and Katakana**

**Description**

Romanize Japanese Hiragana and Katakana

**Usage**

```r
strj_romanize(
  text,
  config = c("wikipedia", "traditional hepburn", "modified hepburn", "kunrei", "nihon")
)
```

**Arguments**

- **text**: Character vector. If elements are composed of except but hiragana and katakana letters, those letters are dropped from the return value.
- **config**: Configuration used to romanize. Default is wikipedia.
Details

There are several ways to romanize Japanese. Using this implementation, you can convert hiragana and katakana as 5 different styles: the wikipedia style, the traditional hepburn style, the modified hepburn style, the kunrei style, and the nihon style.

Note that all of these styles return a slightly different form of stringi::stri_trans_general(text, "Any-latn").

Value

A character vector.

See Also

https://github.com/hakatashi/japanese.js#japaneseromanizetext-config

Examples

```r
strj_romanize(
  paste0(
    "\u3042\u306e\u30a4\u30fc\u30c8",
    "\u30fc\u30f4\u30a9\u306e\u304a",
    "\u3068\u304a\u3063\u98a8"
  )
)
```

---

`strj_segment`  
Segment text into tokens

Description

An alias of strj_tokenize(engine = "budoux").

Usage

```r
strj_segment(text, format = c("list", "data.frame"), split = FALSE)
```

Arguments

- `text`: Character vector to be tokenized.
- `format`: Output format. Choose list or data.frame.
- `split`: Logical. If passed as, the function splits the vector into some sentences using stringi::stri_split_boundaries(type = "sentence") before tokenizing.

Value

A List or a data.frame.
Examples

```r
strj_tinyseg(
  paste0(
    "あのイート",
    "ーヴォのすき",
    "とおった風"
  )
)
)
```

```
strj_tinyseg(
  paste0(
    "あのイート",
    "ーヴォのすき",
    "とおった風"
  ),
  format = "data.frame"
)
```

---

**strj_tinyseg**  
Segment text into phrases

**Description**

An alias of `strj_tokenize(engine = "tinyseg")`.

**Usage**

```r
strj_tinyseg(text, format = c("list", "data.frame"), split = FALSE)
```

**Arguments**

- `text`  
  Character vector to be tokenized.
- `format`  
  Output format. Choose `list` or `data.frame`.
- `split`  
  Logical. If passed as `TRUE`, the function splits vectors into some sentences using `stringi::stri_split_boundaries(type = "sentence")` before tokenizing.

**Value**

A list or a data.frame.

**Examples**

```r
strj_tinyseg(
  paste0(
    "あのイート",
    "ーヴォのすき",
    "とおった風"
  )
)
```
strj_tokenize

Split text into tokens

Description

Splits text into several tokens using specified tokenizer.

Usage

strj_tokenize(
  text,             
  format = c("list", "data.frame"),
  engine = c("stringi", "budoux", "tinyseg", "mecab", "sudachipy"),
  rcpath = NULL,    
  mode = c("C", "B", "A"),
  split = FALSE)

Arguments

text            Character vector to be tokenized.
format          Output format. Choose list or data.frame.
engine          Tokenizer name. Choose one of 'stringi', 'budoux', 'tinyseg', 'mecab', or 'sudachipy'. Note that the specified tokenizer is installed and available when you use 'mecab' or 'sudachipy'.
rcpath          Path to a setting file for 'MeCab' or 'sudachipy' if any.
mode            Splitting mode for 'sudachipy'.
split           Logical. If passed as TRUE, the function splits the vector into some sentences using stringi::stri_split_boundaries(type = "sentence") before tokenizing.

Value

A list or a data.frame.
**Examples**

```r
strj_tokenize(
  paste0(
    "\u3042\u306e\u30a4\u30fc\u30cf\u30c8",
    "\u30fc\u30f4\u30a9\u306e\u3059\u304d",
    "\u3068\u304a\u3063\u305f\u98a8"
  )
)
strj_tokenize(
  paste0(
    "\u3042\u306e\u30a4\u30fc\u30cf\u30c8",
    "\u30fc\u30f4\u30a9\u306e\u3059\u304d",
    "\u3068\u304a\u3063\u305f\u98a8"
  ),
  format = "data.frame"
)
```

---

**strj_transcribe_num**  
*Transcribe Arabic to Kansuji*

**Description**

Transcribes Arabic integers to Kansuji with auxiliary numerals.

**Usage**

```r
strj_transcribe_num(int)
```

**Arguments**

- `int`  
  Integers.

**Details**

As its implementation is limited, this function can only transcribe numbers up to trillions. In case you convert much bigger numbers, try to use the `arabic2kansuji` package.

**Value**

A character vector.

**Examples**

```r
strj_transcribe_num(c(10L, 31415L))
```
Index

* datasets
  hiroba, 6
  polano, 10
bind_lr, 2
bind_tf_idf2, 3
collapse_tokens, 5
get_dict_features, 5
hiroba, 6
lex_density, 7
mute_tokens, 8
ngram_tokenizer, 8
pack, 9
polano, 10
prettify, 10
read_rewrite_def, 11
strj_fill_iter_mark, 12
strj_hiraganize, 12
strj_katakanaize, 13
strj_normalize, 14
strj_rewrite_as_def, 14
strj_romanize, 15
strj_segment, 16
strj_tinyseg, 17
strj_tokenize, 18
strj_transcribe_num, 19