Package 'tokenizers'

December 22, 2022

Type Package

Title Fast, Consistent Tokenization of Natural Language Text

Version 0.3.0

Date 2022-12-19

Description Convert natural language text into tokens. Includes tokenizers for shingled n-grams, skip n-grams, words, word stems, sentences, paragraphs, characters, shingled characters, lines, Penn Treebank, regular expressions, as well as functions for counting characters, words, and sentences, and a function for splitting longer texts into separate documents, each with the same number of words. The tokenizers have a consistent interface, and the package is built on the 'stringi' and 'Rcpp' packages for fast yet correct tokenization in 'UTF-8'.

License MIT + file LICENSE

LazyData yes

URL https://docs.ropensci.org/tokenizers/, https://github.com/ropensci/tokenizers

BugReports https://github.com/ropensci/tokenizers/issues

RoxygenNote 7.2.1

Depends R (>= 3.1.3)

Imports stringi (>= 1.0.1), Rcpp (>= 0.12.3), SnowballC (>= 0.5.1)

LinkingTo Rcpp

Encoding UTF-8

Suggests covr, knitr, rmarkdown, stopwords (>= 0.9.0), testthat

VignetteBuilder knitr

NeedsCompilation yes

Author Lincoln Mullen [aut, cre] (<https://orcid.org/0000-0001-5103-6917>), Os Keyes [ctb] (<https://orcid.org/0000-0001-5196-609X>), Dmitriy Selivanov [ctb], Jeffrey Arnold [ctb] (<https://orcid.org/0000-0001-9953-3904>), Kenneth Benoit [ctb] (<https://orcid.org/0000-0002-0797-564X>)

Maintainer Lincoln Mullen <lincoln@lincolnmullen.com>

Repository CRAN

Date/Publication 2022-12-22 08:50:02 UTC

R topics documented:

`basic-tokenizers` *Basic tokenizers*

Description

These functions perform basic tokenization into words, sentences, paragraphs, lines, and characters. The functions can be piped into one another to create at most two levels of tokenization. For instance, one might split a text into paragraphs and then word tokens, or into sentences and then word tokens.

Usage

```
tokenize_characters(
  x,
  lowercase = TRUE,
  strip_non_alphanum = TRUE,
  simplify = FALSE
)

tokenize_words(
  x,
  lowercase = TRUE,
  stopwords = NULL,
  strip_punct = TRUE,
  strip_numeric = FALSE,
  simplify = FALSE
)

tokenize_sentences(x, lowercase = FALSE, strip_punct = FALSE, simplify = FALSE)
```

```
tokenize_lines(x, simplify = FALSE)

tokenize_paragraphs(x, paragraph_break = "\n\n", simplify = FALSE)

tokenize_regex(x, pattern = "\\s+", simplify = FALSE)
```

Arguments

x	A character vector or a list of character vectors to be tokenized. If x is a character vector, it can be of any length, and each element will be tokenized separately. If x is a list of character vectors, where each element of the list should have a length of 1.
lowercase	Should the tokens be made lower case? The default value varies by tokenizer; it is only TRUE by default for the tokenizers that you are likely to use last.
strip_non_alphanum	Should punctuation and white space be stripped?
simplify	FALSE by default so that a consistent value is returned regardless of length of input. If TRUE, then an input with a single element will return a character vector of tokens instead of a list.
stopwords	A character vector of stop words to be excluded.
strip_punct	Should punctuation be stripped?
strip_numeric	Should numbers be stripped?
paragraph_break	A string identifying the boundary between two paragraphs.
pattern	A regular expression that defines the split.

Value

A list of character vectors containing the tokens, with one element in the list for each element that was passed as input. If simplify = TRUE and only a single element was passed as input, then the output is a character vector of tokens.

Examples

```
song <-  paste0("How many roads must a man walk down\n",
                "Before you call him a man?\n",
                "How many seas must a white dove sail\n",
                "Before she sleeps in the sand?\n",
                "\n",
                "How many times must the cannonballs fly\n",
                "Before they're forever banned?\n",
                "The answer, my friend, is blowin' in the wind.\n",
                "The answer is blowin' in the wind.\n")

tokenize_words(song)
tokenize_words(song, strip_punct = FALSE)
tokenize_sentences(song)
```

```
tokenize_paragraphs(song)
tokenize_lines(song)
tokenize_characters(song)
```

---

chunk_text *Chunk text into smaller segments*

Description

Given a text or vector/list of texts, break the texts into smaller segments each with the same number of words. This allows you to treat a very long document, such as a novel, as a set of smaller documents.

Usage

```
chunk_text(x, chunk_size = 100, doc_id = names(x), ...)
```

Arguments

x	A character vector or a list of character vectors to be tokenized into n-grams. If x is a character vector, it can be of any length, and each element will be chunked separately. If x is a list of character vectors, each element of the list should have a length of 1.
chunk_size	The number of words in each chunk.
doc_id	The document IDs as a character vector. This will be taken from the names of the x vector if available. NULL is acceptable.
...	Arguments passed on to tokenize_words.

Details

Chunking the text passes it through tokenize_words, which will strip punctuation and lowercase the text unless you provide arguments to pass along to that function.

Examples

```
## Not run:
chunked <- chunk_text(mobydick, chunk_size = 100)
length(chunked)
chunked[1:3]

## End(Not run)
```

count_words *Count words, sentences, characters*

Description

Count words, sentences, and characters in input texts. These functions use the `stringi` package, so they handle the counting of Unicode strings (e.g., characters with diacritical marks) in a way that makes sense to people counting characters.

Usage

```
count_words(x)

count_characters(x)

count_sentences(x)
```

Arguments

`x`	A character vector or a list of character vectors. If `x` is a character vector, it can be of any length, and each element will be tokenized separately. If `x` is a list of character vectors, each element of the list should have a length of 1.

Value

An integer vector containing the counted elements. If the input vector or list has names, they will be preserved.

Examples

```
count_words(mobydick)
count_sentences(mobydick)
count_characters(mobydick)
```

mobydick *The text of Moby Dick*

Description

The text of Moby Dick, by Herman Melville, taken from Project Gutenberg.

Usage

```
mobydick
```

Format

A named character vector with length 1.

Source

http://www.gutenberg.org/

`ngram-tokenizers` *N-gram tokenizers*

Description

These functions tokenize their inputs into different kinds of n-grams. The input can be a character vector of any length, or a list of character vectors where each character vector in the list has a length of 1. See details for an explanation of what each function does.

Usage

```
tokenize_ngrams(
  x,
  lowercase = TRUE,
  n = 3L,
  n_min = n,
  stopwords = character(),
  ngram_delim = " ",
  simplify = FALSE
)

tokenize_skip_ngrams(
  x,
  lowercase = TRUE,
  n_min = 1,
  n = 3,
  k = 1,
  stopwords = character(),
  simplify = FALSE
)
```

Arguments

`x`	A character vector or a list of character vectors to be tokenized into n-grams. If `x` is a character vector, it can be of any length, and each element will be tokenized separately. If `x` is a list of character vectors, each element of the list should have a length of 1.
`lowercase`	Should the tokens be made lower case?
`n`	The number of words in the n-gram. This must be an integer greater than or equal to 1.

n_min	The minimum number of words in the n-gram. This must be an integer greater than or equal to 1, and less than or equal to n.
stopwords	A character vector of stop words to be excluded from the n-grams.
ngram_delim	The separator between words in an n-gram.
simplify	FALSE by default so that a consistent value is returned regardless of length of input. If TRUE, then an input with a single element will return a character vector of tokens instead of a list.
k	For the skip n-gram tokenizer, the maximum skip distance between words. The function will compute all skip n-grams between 0 and k.

Details

tokenize_ngrams: Basic shingled n-grams. A contiguous subsequence of n words. This will compute shingled n-grams for every value of between n_min (which must be at least 1) and n.

tokenize_skip_ngrams: Skip n-grams. A subsequence of n words which are at most a gap of k words between them. The skip n-grams will be calculated for all values from 0 to k.

These functions will strip all punctuation and normalize all whitespace to a single space character.

Value

A list of character vectors containing the tokens, with one element in the list for each element that was passed as input. If simplify = TRUE and only a single element was passed as input, then the output is a character vector of tokens.

Examples

```
song <-  paste0("How many roads must a man walk down\n",
                "Before you call him a man?\n",
                "How many seas must a white dove sail\n",
                "Before she sleeps in the sand?\n",
                "\n",
                "How many times must the cannonballs fly\n",
                "Before they're forever banned?\n",
                "The answer, my friend, is blowin' in the wind.\n",
                "The answer is blowin' in the wind.\n")

tokenize_ngrams(song, n = 4)
tokenize_ngrams(song, n = 4, n_min = 1)
tokenize_skip_ngrams(song, n = 4, k = 2)
```

---

tokenizers *Tokenizers*

Description

A collection of functions with a consistent interface to convert natural language text into tokens.

Details

The tokenizers in this package have a consistent interface. They all take either a character vector of any length, or a list where each element is a character vector of length one. The idea is that each element comprises a text. Then each function returns a list with the same length as the input vector, where each element in the list are the tokens generated by the function. If the input character vector or list is named, then the names are preserved.

`tokenize_character_shingles` *Character shingle tokenizers*

Description

The character shingle tokenizer functions like an n-gram tokenizer, except the units that are shingled are characters instead of words. Options to the function let you determine whether non-alphanumeric characters like punctuation should be retained or discarded.

Usage

```
tokenize_character_shingles(
  x,
  n = 3L,
  n_min = n,
  lowercase = TRUE,
  strip_non_alphanum = TRUE,
  simplify = FALSE
)
```

Arguments

`x`	A character vector or a list of character vectors to be tokenized into character shingles. If `x` is a character vector, it can be of any length, and each element will be tokenized separately. If `x` is a list of character vectors, each element of the list should have a length of 1.
`n`	The number of characters in each shingle. This must be an integer greater than or equal to 1.
`n_min`	This must be an integer greater than or equal to 1, and less than or equal to `n`.
`lowercase`	Should the characters be made lower case?
`strip_non_alphanum`	Should punctuation and white space be stripped?
`simplify`	`FALSE` by default so that a consistent value is returned regardless of length of input. If `TRUE`, then an input with a single element will return a character vector of tokens instead of a list.

Value

A list of character vectors containing the tokens, with one element in the list for each element that was passed as input. If `simplify = TRUE` and only a single element was passed as input, then the output is a character vector of tokens.

Examples

```
x <- c("Now is the hour of our discontent")
tokenize_character_shingles(x)
tokenize_character_shingles(x, n = 5)
tokenize_character_shingles(x, n = 5, strip_non_alphanum = FALSE)
tokenize_character_shingles(x, n = 5, n_min = 3, strip_non_alphanum = FALSE)
```

---

tokenize_ptb *Penn Treebank Tokenizer*

Description

This function implements the Penn Treebank word tokenizer.

Usage

```
tokenize_ptb(x, lowercase = FALSE, simplify = FALSE)
```

Arguments

`x`	A character vector or a list of character vectors to be tokenized into n-grams. If `x` is a character vector, it can be of any length, and each element will be tokenized separately. If `x` is a list of character vectors, each element of the list should have a length of 1.
`lowercase`	Should the tokens be made lower case?
`simplify`	`FALSE` by default so that a consistent value is returned regardless of length of input. If `TRUE`, then an input with a single element will return a character vector of tokens instead of a list.

Details

This tokenizer uses regular expressions to tokenize text similar to the tokenization used in the Penn Treebank. It assumes that text has already been split into sentences. The tokenizer does the following:

- splits common English contractions, e.g. `don't` is tokenized into `do n't` and `they'll` is tokenized into -> `they 'll`,
- handles punctuation characters as separate tokens,
- splits commas and single quotes off from words, when they are followed by whitespace,
- splits off periods that occur at the end of the sentence.

This function is a port of the Python NLTK version of the Penn Treebank Tokenizer.

Value

A list of character vectors containing the tokens, with one element in the list for each element that was passed as input. If `simplify = TRUE` and only a single element was passed as input, then the output is a character vector of tokens.

References

NLTK TreebankWordTokenizer

Examples

```
song <- list(paste0("How many roads must a man walk down\n",
                    "Before you call him a man?"),
             paste0("How many seas must a white dove sail\n",
                    "Before she sleeps in the sand?\n"),
             paste0("How many times must the cannonballs fly\n",
                    "Before they're forever banned?\n"),
             "The answer, my friend, is blowin' in the wind.",
             "The answer is blowin' in the wind.")
tokenize_ptb(song)
tokenize_ptb(c("Good muffins cost $3.88\nin New York. Please buy me\ntwo of them.",
  "They'll save and invest more.",
  "Hi, I can't say hello."))
```

---

tokenize_word_stems *Word stem tokenizer*

Description

This function turns its input into a character vector of word stems. This is just a wrapper around the `wordStem` function from the SnowballC package which does the heavy lifting, but this function provides a consistent interface with the rest of the tokenizers in this package. The input can be a character vector of any length, or a list of character vectors where each character vector in the list has a length of 1.

Usage

```
tokenize_word_stems(
  x,
  language = "english",
  stopwords = NULL,
  simplify = FALSE
)
```

Arguments

x	A character vector or a list of character vectors to be tokenized. If x is a character vector, it can be of any length, and each element will be tokenized separately. If x is a list of character vectors, where each element of the list should have a length of 1.
language	The language to use for word stemming. This must be one of the languages available in the SnowballC package. A list is provided by getStemLanguages.
stopwords	A character vector of stop words to be excluded
simplify	FALSE by default so that a consistent value is returned regardless of length of input. If TRUE, then an input with a single element will return a character vector of tokens instead of a list.

Details

This function will strip all white space and punctuation and make all word stems lowercase.

Value

A list of character vectors containing the tokens, with one element in the list for each element that was passed as input. If simplify = TRUE and only a single element was passed as input, then the output is a character vector of tokens.

See Also

wordStem

Examples

```
song <-  paste0("How many roads must a man walk down\n",
                "Before you call him a man?\n",
                "How many seas must a white dove sail\n",
                "Before she sleeps in the sand?\n",
                "\n",
                "How many times must the cannonballs fly\n",
                "Before they're forever banned?\n",
                "The answer, my friend, is blowin' in the wind.\n",
                "The answer is blowin' in the wind.\n")

tokenize_word_stems(song)
```

Index