Package 'ngram'

May 9, 2026

Type Package

Title Fast n-Gram 'Tokenization'

Version 3.2.3

Description An n-gram is a sequence of n ``words'' taken, in order, from a body of text. This is a collection of utilities for creating, displaying, summarizing, and ``babbling'' n-grams. The 'tokenization' and ``babbling'' are handled by very efficient C code, which can even be built as its own standalone library. The babbler is a simple Markov chain. The package also offers a vignette with complete example 'workflows' and information about the utilities offered in the package.

License BSD 2-clause License + file LICENSE

Depends R (>= 3.0.0)

Imports methods

LazyLoad yes

NeedsCompilation yes

ByteCompile yes

Maintainer Drew Schmidt <wrathematics@gmail.com>

URL https://github.com/wrathematics/ngram

BugReports https://github.com/wrathematics/ngram/issues

RoxygenNote 7.1.2

Author Drew Schmidt [aut, cre],
Christian Heckendorf [aut]

Repository CRAN

Date/Publication 2023-12-10 19:00:02 UTC

Contents

`ngram-package` *ngram: Fast n-Gram Tokenization*

Description

An n-gram is a sequence of n "words" taken from a body of text. This package offers utilities for creating, displaying, summarizing, and "babbling" n-grams. The tokenization and "babbling" are handled by very efficient C code, which can even be build as its own standalone library. The babbler is a simple Markov chain.

Details

The ngram package is distributed under the permissive 2-clause BSD license. If you find the code here useful, please let us know and/or cite the package, whatever is appropriate.

The package has its own PRNG; we use an implementation of MT1997 for all non-deterministic choices.

`babble` *ngram Babbler*

Description

The babbler uses its own internal PRNG (i.e., not R's), so seeds cannot be managed as with R's seeds. The generator is an implementation of MT19937.

At this time, we note that the seed may not guarantee the same results across machines. Currently only Solaris produces different values from mainstream platforms (Windows, Mac, Linux, FreeBSD), but potentially others could as well.

Usage

```
babble(ng, genlen = 150, seed = getseed())

## S4 method for signature 'ngram'
babble(ng, genlen = 150, seed = getseed())
```

Arguments

ng	An ngram object.
genlen	Generated length, i.e., the number of words to babble.
seed	Seed for the random number generator.

Details

A markov chain babbler.

See Also

ngram, getseed

Examples

```
library(ngram)

str = "A B A C A B B"
ng = ngram(str)
babble(ng, genlen=5, seed=1234)
```

---

concatenate *Concatenate*

Description

A quick utility for concatenating strings together. This is handy because if you want to generate the n-grams for several different texts, you must first put them into a single string unless the text is composed of sentences that should not be joined.

Usage

```
concatenate(..., collapse = " ", rm.space = FALSE)
```

Arguments

...	Input text(s).
collapse	A character to separate the input strings if a vector of strings is supplied; otherwise this does nothing.
rm.space	logical; determines if spaces should be removed from the final string.

Value

A string.

See Also

preprocess

Examples

```
library(ngram)

words = c("a", "b", "c")
wordcount(words)
str = concatenate(words)
wordcount(str)
```

---

getseed *getseed*

Description

A seed generator for use with the ngram package.

Usage

```
getseed()
```

Details

Uses a 96-bit hash of the current process id, time, and a random uniform value from R's random generator.

See Also

babble

getters *ngram Getters*

Description

Some simple "getters" for ngram objects. Necessary since the internal representation is not a native R object.

Usage

```
ng_order(ng, decreasing = FALSE)

## S4 method for signature 'ngram'
ng_order(ng, decreasing = FALSE)

get.ngrams(ng)

## S4 method for signature 'ngram'
get.ngrams(ng)

get.string(ng)

## S4 method for signature 'ngram'
get.string(ng)

get.nextwords(ng)

## S4 method for signature 'ngram'
get.nextwords(ng)
```

Arguments

ng	An ngram object.
decreasing	Should the sorted order be in descending order?

Details

ngram.order returns an R vector with the original corpus order of the ngrams.

get.ngrams() returns an R vector of all n-grams.

get.nextwords() does nothing at the moment; it will be implemented in future releases.

getnstring() recovers the input string as an R string.

See Also

ngram-class, ngram

Examples

```
library(ngram)

str = "A B A C A B B"
ng = ngram(str)
get.ngrams(ng)[ng_order(ng)]
```

`multiread` *Multiread*

Description

Read in a collection of text files.

Usage

```
multiread(
  path = ".",
  extension = "txt",
  recursive = FALSE,
  ignore.case = FALSE,
  prune.empty = TRUE,
  pathnames = TRUE
)
```

Arguments

`path`	The base file path to search.
`extension`	An extension or the "*" wildcard (for everything). For example, to read in files ending `.txt`, you could specify `extension="txt"`. For the purposes of this function, each of `*.txt`, `*txt`, `.txt`, and `txt` are treated the same.
`recursive`	Logical; should the search include all subdirectories?
`ignore.case`	Logical; should case be ignored in the extension? For example, if `TRUE`, then `.r` and `.R` files are treated the same.
`prune.empty`	Logical; should empty files be removed from the returned list?
`pathnames`	Logical; should the full path be included in the names of the returned list.

Details

The `extension` argument is not a general regular expression pattern, but a simplified pattern. For example, the pattern `*.txt` is really equivalent to `*[.]txt$` as a regular expression. If you need more complicated patterns, you should directly use the `dir()` function.

Value

A named list of strings, where the names are the file names.

Examples

```
## Not run:
path = system.file(package="ngram")

### Read all files in the base path
multiread(path, extension="*")

### Read all .r/.R files recursively (warning: lots of text)
multiread(path, extension="r", recursive=TRUE, ignore.case=TRUE)


## End(Not run)
```

---

ngram *n-gram Tokenization*

Description

The ngram() function is the main workhorse of this package. It takes an input string and converts it into the internal n-gram representation.

Usage

```
ngram(str, n = 2, sep = " ")
```

Arguments

str	The input text.
n	The 'n' as in 'n-gram'.
sep	A set of separator characters for the "words". See details for information about how this works; it works a little differently from sep arguments in R functions.

Details

On evaluation, a copy of the input string is produced and stored as an external pointer. This is necessary because the internal list representation just points to the first char of each word in the input string. So if you (or R's gc) deletes the input string, basically all hell breaks loose.

The sep parameter splits at any of the characters in the string. So sep=", " splits at a comma or a space.

Value

An ngram class object.

See Also

ngram-class, getters, phrasetable, babble

Examples

```
library(ngram)

str = "A B A C A B B"
ngram(str, n=2)

str = "A,B,A,C A B B"
### Split at a space
print(ngram(str), output="full")
### Split at a comma
print(ngram(str, sep=","), output="full")
### Split at a space or a comma
print(ngram(str, sep=", "), output="full")
```

---

ngram-class *Class ngram*

Description

An n-gram is an ordered sequence of n "words" taken from a body of "text". The terms "words" and "text" can easily be interpreted literally, or with a more loose interpretation.

Details

For example, consider the sequence "A B A C A B B". If we examine the 2-grams (or bigrams) of this sequence, they are

A B, B A, A C, C A, A B, B B

or without repetition:

A B, B A, A C, C A, B B

That is, we take the input string and group the "words" 2 at a time (because `n=2`). Notice that the number of n-grams and the number of words are not obviously related; counting repetition, the number of n-grams is equal to

`nwords - n + 1`

Bounds ignoring repetition are highly dependent on the input. A correct but useless bound is

`\#ngrams = nwords - (\#repeats - 1) - (n - 1)`

An `ngram` object is an S4 class container that stores some basic summary information (e.g., n), and several external pointers. For information on how to construct an `ngram` object, see `ngram`.

Slots

`str_ptr` A pointer to a copy of the original input string.

`strlen` The length of the string.

`n` The eponymous 'n' as in 'n-gram'.

`ngl_ptr` A pointer to the processed list of n-grams.

`ngsize` The length of the ngram list, or in other words, the number of unique n-grams in the input string.

`sl_ptr` A pointer to the list of words from the input string.

See Also

`ngram`

`ngram-print` *ngram printing*

Description

Print methods.

Usage

```
## S4 method for signature 'ngram'
print(x, output = "summary")

## S4 method for signature 'ngram'
show(object)
```

Arguments

`x, object`	An ngram object.
`output`	a character string; determines what exactly is printed. Options are "summary", "truncated", and "full".

Details

If `output=="summary"`, then just a simple representation of the n-gram object will be printed; for example, "An ngram object with 5 2-grams".

If `output=="truncated"`, then the n-grams will be printed up to a maximum of 5 total.

If `output=="full"` then all n-grams will be printed.

See Also

`ngram`, `babble`

phrasetable *Get Phrasetable*

Description

Get a table

Usage

```
get.phrasetable(ng)
```

Arguments

ng	An ngram object.

See Also

ngram-class

Examples

```
library(ngram)

str = "A B A C A B B"
ng = ngram(str)
get.phrasetable(ng)
```

preprocess *Basic Text Preprocessor*

Description

A simple text preprocessor for use with the `ngram()` function.

Usage

```
preprocess(
  x,
  case = "lower",
  remove.punct = FALSE,
  remove.numbers = FALSE,
  fix.spacing = TRUE
)
```

Arguments

x	Input text.
case	Option to change the case of the text. Value should be "upper", "lower", or NULL (no change).
remove.punct	Logical; should punctuation be removed?
remove.numbers	Logical; should numbers be removed?
fix.spacing	Logical; should multi/trailing spaces be collapsed/removed.

Details

The input text x must already be in the correct form for ngram(), i.e., a single string (character vector of length 1).

The case argument can take 3 possible values: NULL, in which case nothing is done, or lower or upper, wherein the case of the input text will be made lower/upper case, repesctively.

Value

concat() returns

Examples

```
library(ngram)

x = "Watch  out    for snakes!  111"
preprocess(x)
preprocess(x, remove.punct=TRUE, remove.numbers=TRUE)
```

---

rcorpus *Random Corpus*

Description

Generate a corpus of random "words".

Usage

```
rcorpus(nwords = 50, alphabet = letters, minwordlen = 1, maxwordlen = 6)
```

Arguments

nwords	Number of words to generate.
alphabet	The pool of "letters" that word generation coes from. By default, it is the lowercase roman alphabet.
minwordlen, maxwordlen	The min/max length of words in the generated corpus.

Value

A string.

Examples

```
rcorpus(10)
```

`splitter` *Character Splitter*

Description

A utility function for use with n-gram modeling. This function splits a string based on various options.

Usage

```
splitter(
  string,
  split.char = FALSE,
  split.space = TRUE,
  spacesep = "_",
  split.punct = FALSE
)
```

Arguments

`string`	An input string.
`split.char`	Logical; should a split occur after every character?
`split.space`	Logical; determines if spaces should be preserved as characters in the n-gram tokenization. The character(s) used for spaces are determined by the `spacesep` argument. characters.
`spacesep`	The character(s) to represent a space in the case that `split.space=TRUE`. Should not just be a space(s).
`split.punct`	Logical; determines if splits should occur at punctuation.

Details

Note that choosing `split.char=TRUE` necessarily implies `split.punct=TRUE` as well — but *not* necessarily that `split.space=TRUE`.

Value

A string.

Examples

```
x = "watch out! a snake!"

splitter(x, split.char=TRUE)
splitter(x, split.space=TRUE, spacesep="_")
splitter(x, split.punct=TRUE)
```

`string.summary` *Text Summary*

Description

Text Summary

Usage

```
string.summary(string, wordlen_max = 10, senlen_max = 10, syllen_max = 10)
```

Arguments

`string`	An input string.
`wordlen_max`, `senlen_max`, `syllen_max`	The maximum lengths of words/sentences/syllables to consider.

Value

A list of class `string_summary`.

Examples

```
x = "a b a c a b b"

string.summary(x)
```

Tokenize-AsWeka *Weka-like n-gram Tokenization*

Description

An n-gram tokenizer with identical output to the NGramTokenizer function from the RWeka package.

Usage

```
ngram_asweka(str, min = 2, max = 2, sep = " ")
```

Arguments

str	The input text.
min, max	The minimum and maximum 'n' as in 'n-gram'.
sep	A set of separator characters for the "words". See details for information about how this works; it works a little differently from sep arguments in R functions.

Details

This n-gram tokenizer behaves similarly in both input and return to the tokenizer in RWeka. Unlike the tokenizer ngram(), the return is not a special class of external pointers; it is a vector, and therefore can be serialized via save() or saveRDS().

Value

A vector of n-grams listed in decreasing blocks of n, in order within a block. The output matches that of RWeka's n-gram tokenizer.

See Also

ngram

Examples

```
library(ngram)

str = "A B A C A B B"
ngram_asweka(str, min=2, max=4)
```

wordcount *wordcount*

Description

wordcount() counts words. Currently a "word" is a clustering of characters separated from another clustering of charactersby at least 1 space. That is the law.

Usage

```
wordcount(x, sep = " ", count_fun = sum)

## S3 method for class 'character'
wordcount(x, sep = " ", count_fun = sum)

## S3 method for class 'ngram'
wordcount(x, sep = " ", count_fun = sum)
```

Arguments

x	A string or vector of strings, or an ngram object.
sep	The characters used to separate words.
count_fun	The function to use for aggregation if x has length greater than 1. Useful ones include sum and identity.

Value

A count.

See Also

preprocess

Examples

```
library(ngram)

words = c("a", "b", "c")
words
wordcount(words)

str = concatenate(words, collapse="")
str
wordcount(str)
```

Index