Package: textreuse (via r-universe)

August 29, 2024

Type Package

Title Detect Text Reuse and Document Similarity

Version 0.1.5

Date 2024-05-24

Description Tools for measuring similarity among documents and detecting passages which have been reused. Implements shingled n-gram, skip n-gram, and other tokenizers; similarity/dissimilarity functions; pairwise comparisons; minhash and locality sensitive hashing algorithms; and a version of the Smith-Waterman local alignment algorithm suitable for natural language.

License MIT

LazyData TRUE

URL https://docs.ropensci.org/textreuse (website)
https://github.com/ropensci/textreuse

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

VignetteBuilder knitr

Depends R (>= 3.1.1)

Imports assertthat (>= 0.1), digest (>= 0.6.8), dplyr (>= 0.8.0), NLP (>= 0.1.8), Rcpp (>= 0.12.0), RcppProgress (>= 0.1), stringr (>= 1.0.0), tibble (>= 3.0.1), tidyr (>= 0.3.1)

Suggests testthat (>= 0.11.0), knitr (>= 1.11), rmarkdown (>= 0.8), covr

LinkingTo BH, Rcpp, RcppProgress

RoxygenNote 7.3.1

Encoding UTF-8

Repository https://ropensci.r-universe.dev

RemoteUrl https://github.com/ropensci/textreuse

RemoteRef master

RemoteSha 9cf2568ee8a758aac30c8f64edcfc6f7c17e1434

Contents

`textreuse-package` *textreuse: Detect Text Reuse and Document Similarity*

Description

Tools for measuring similarity among documents and detecting passages which have been reused. Implements shingled n-gram, skip n-gram, and other tokenizers; similarity/dissimilarity functions; pairwise comparisons; minhash and locality sensitive hashing algorithms; and a version of the Smith-Waterman local alignment algorithm suitable for natural language.

Details

The best place to begin with this package in the introductory vignette.

```
vignette("textreuse-introduction", package = "textreuse")
```

After reading that vignette, the "pairwise" and "minhash" vignettes introduce specific paths for working with the package.

```
vignette("textreuse-pairwise", package = "textreuse")
vignette("textreuse-minhash", package = "textreuse")
vignette("textreuse-alignment", package = "textreuse")
```

Another good place to begin with the package is the documentation for loading documents (`TextReuseTextDocument` and `TextReuseCorpus`), for tokenizers, similarity functions, and locality-sensitive hashing.

Author(s)

Maintainer: Yaoxiang Li <liyaoxiang@outlook.com> (ORCID)

Authors:

- Lincoln Mullen <lincoln@lincolnmullen.com> (ORCID)

References

The sample data provided in the `extdata/legal` directory is taken from a corpus of American Tract Society publications from the nineteen-century, gathered from the Internet Archive.

The sample data provided in the `extdata/legal` directory, are taken from the following nineteenth-century codes of civil procedure from California and New York.

Final Report of the Commissioners on Practice and Pleadings, in 2 *Documents of the Assembly of New York*, 73rd Sess., No. 16, (1850): 243-250, sections 597-613. Google Books.

An Act To Regulate Proceedings in Civil Cases, 1851 *California Laws* 51, 51-53 sections 4-17; 101, sections 313-316. Google Books.

See Also

Useful links:

- https://docs.ropensci.org/textreuse
- https://github.com/ropensci/textreuse
- Report bugs at https://github.com/ropensci/textreuse/issues

---

`align_local` *Local alignment of natural language texts*

Description

This function takes two texts, either as strings or as `TextReuseTextDocument` objects, and finds the optimal local alignment of those texts. A local alignment finds the best matching subset of the two documents. This function adapts the Smith-Waterman algorithm, used for genetic sequencing, for use with natural language. It compare the texts word by word (the comparison is case-insensitive) and scores them according to a set of parameters. These parameters define the score for a `match`, and the penalties for a `mismatch` and for opening a `gap` (i.e., the first mismatch in a potential sequence). The function then reports the optimal local alignment. Only the subset of the documents that is a match is included. Insertions or deletions in the text are reported with the `edit_mark` character.

Usage

```
align_local(
  a,
  b,
  match = 2L,
  mismatch = -1L,
  gap = -1L,
  edit_mark = "#",
  progress = interactive()
)
```

Arguments

a	A character vector of length one, or a TextReuseTextDocument.
b	A character vector of length one, or a TextReuseTextDocument.
match	The score to assign a matching word. Should be a positive integer.
mismatch	The score to assign a mismatching word. Should be a negative integer or zero.
gap	The penalty for opening a gap in the sequence. Should be a negative integer or zero.
edit_mark	A single character used for displaying for displaying insertions/deletions in the documents.
progress	Display a progress bar and messages while computing the alignment.

Details

The compute time of this function is proportional to the product of the lengths of the two documents. Thus, longer documents will take considerably more time to compute. This function has been tested with pairs of documents containing about 25 thousand words each.

If the function reports that there were multiple optimal alignments, then it is likely that there is no strong match in the document.

The score reported for the local alignment is dependent on both the size of the documents and on the strength of the match, as well as on the parameters for match, mismatch, and gap penalties, so the scores are not directly comparable.

Value

A list with the class textreuse_alignment. This list contains several elements:

- a_edit and b_edit: Character vectors of the sequences with edits marked.
- score: The score of the optimal alignment.

References

For a useful description of the algorithm, see this post. For the application of the Smith-Waterman algorithm to natural language, see David A. Smith, Ryan Cordell, and Elizabeth Maddock Dillon, "Infectious Texts: Modeling Text Reuse in Nineteenth-Century Newspapers." IEEE International Conference on Big Data, 2013, http://hdl.handle.net/2047/d20004858.

Examples

```
align_local("The answer is blowin' in the wind.",
            "As the Bob Dylan song says, the answer is blowing in the wind.")

# Example of matching documents from a corpus
dir <- system.file("extdata/legal", package = "textreuse")
corpus <- TextReuseCorpus(dir = dir, progress = FALSE)
alignment <- align_local(corpus[["ca1851-match"]], corpus[["ny1850-match"]])
str(alignment)
```

---

as.matrix.textreuse_candidates *Convert candidates data frames to other formats*

Description

These S3 methods convert a `textreuse_candidates` object to a matrix.

Usage

```
## S3 method for class 'textreuse_candidates'
as.matrix(x, ...)
```

Arguments

`x`	An object of class `textreuse_candidates`.
`...`	Additional arguments.

Value

A similarity matrix with row and column names containing document IDs.

---

filenames *Filenames from paths*

Description

This function takes a character vector of paths and returns just the file name, by default without the extension. A `TextReuseCorpus` uses the paths to the files in the corpus as the names of the list. This function is intended to turn those paths into more manageable identifiers.

Usage

```
filenames(paths, extension = FALSE)
```

Arguments

paths	A character vector of paths.
extension	Should the file extension be preserved?

See Also

basename

Examples

```
paths <- c("corpus/one.txt", "corpus/two.md", "corpus/three.text")
filenames(paths)
filenames(paths, extension = TRUE)
```

---

hash_string *Hash a string to an integer*

Description

Hash a string to an integer

Usage

```
hash_string(x)
```

Arguments

x	A character vector to be hashed.

Value

A vector of integer hashes.

Examples

```
s <- c("How", "many", "roads", "must", "a", "man", "walk", "down")
hash_string(s)
```

lsh *Locality sensitive hashing for minhash*

Description

Locality sensitive hashing (LSH) discovers potential matches among a corpus of documents quickly, so that only likely pairs can be compared.

Usage

```
lsh(x, bands, progress = interactive())
```

Arguments

x	A TextReuseCorpus or TextReuseTextDocument.
bands	The number of bands to use for locality sensitive hashing. The number of hashes in the documents in the corpus must be evenly divisible by the number of bands. See lsh_threshold and lsh_probability for guidance in selecting the number of bands and hashes.
progress	Display a progress bar while comparing documents.

Details

Locality sensitive hashing is a technique for detecting document similarity that does not require pairwise comparisons. When comparing pairs of documents, the number of pairs grows rapidly, so that only the smallest corpora can be compared pairwise in a reasonable amount of computation time. Locality sensitive hashing, on the other hand, takes a document which has been tokenized and hashed using a minhash algorithm. (See minhash_generator.) Each set of minhash signatures is then broken into bands comprised of a certain number of rows. (For example, 200 minhash signatures might be broken down into 20 bands each containing 10 rows.) Each band is then hashed to a bucket. Documents with identical rows in a band will be hashed to the same bucket. The likelihood that a document will be marked as a potential duplicate is proportional to the number of bands and inversely proportional to the number of rows in each band.

This function returns a data frame with the additional class lsh_buckets. The LSH technique only requires that the signatures for each document be calculated once. So it is possible, as long as one uses the same minhash function and the same number of bands, to combine the outputs from this function at different times. The output can thus be treated as a kind of cache of LSH signatures.

To extract pairs of documents from the output of this function, see lsh_candidates.

Value

A data frame (with the additional class lsh_buckets), containing a column with the document IDs and a column with their LSH signatures, or buckets.

References

Jure Leskovec, Anand Rajaraman, and Jeff Ullman, *Mining of Massive Datasets* (Cambridge University Press, 2011), ch. 3. See also Matthew Casperson, "Minhash for Dummies" (November 14, 2013).

See Also

minhash_generator, lsh_candidates, lsh_query, lsh_probability, lsh_threshold

Examples

```
dir <- system.file("extdata/legal", package = "textreuse")
minhash <- minhash_generator(200, seed = 235)
corpus <- TextReuseCorpus(dir = dir,
                          tokenizer = tokenize_ngrams, n = 5,
                          minhash_func = minhash)
buckets <- lsh(corpus, bands = 50)
buckets
```

---

lsh_candidates *Candidate pairs from LSH comparisons*

Description

Given a data frame of LSH buckets returned from lsh, this function returns the potential candidates.

Usage

```
lsh_candidates(buckets)
```

Arguments

buckets	A data frame returned from lsh.

Value

A data frame of candidate pairs.

Examples

```
dir <- system.file("extdata/legal", package = "textreuse")
minhash <- minhash_generator(200, seed = 234)
corpus <- TextReuseCorpus(dir = dir,
                          tokenizer = tokenize_ngrams, n = 5,
                          minhash_func = minhash)
buckets <- lsh(corpus, bands = 50)
lsh_candidates(buckets)
```

lsh_compare — *Compare candidates identified by LSH*

Description

The lsh_candidates only identifies potential matches, but cannot estimate the actual similarity of the documents. This function takes a data frame returned by lsh_candidates and applies a comparison function to each of the documents in a corpus, thereby calculating the document similarity score. Note that since your corpus will have minhash signatures rather than hashes for the tokens itself, you will probably wish to use tokenize to calculate new hashes. This can be done for just the potentially similar documents. See the package vignettes for details.

Usage

```
lsh_compare(candidates, corpus, f, progress = interactive())
```

Arguments

candidates	A data frame returned by lsh_candidates.
corpus	The same TextReuseCorpus corpus which was used to generate the candidates.
f	A comparison function such as jaccard_similarity.
progress	Display a progress bar while comparing documents.

Value

A data frame with values calculated for score.

Examples

```
dir <- system.file("extdata/legal", package = "textreuse")
minhash <- minhash_generator(200, seed = 234)
corpus <- TextReuseCorpus(dir = dir,
                          tokenizer = tokenize_ngrams, n = 5,
                          minhash_func = minhash)
buckets <- lsh(corpus, bands = 50)
candidates <- lsh_candidates(buckets)
lsh_compare(candidates, corpus, jaccard_similarity)
```

`lsh_probability` *Probability that a candidate pair will be detected with LSH*

Description

Functions to help choose the correct parameters for the `lsh` and `minhash_generator` functions. Use `lsh_threshold` to determine the minimum Jaccard similarity for two documents for them to likely be considered a match. Use `lsh_probability` to determine the probability that a pair of documents with a known Jaccard similarity will be detected.

Usage

```
lsh_probability(h, b, s)

lsh_threshold(h, b)
```

Arguments

`h`	The number of minhash signatures.
`b`	The number of LSH bands.
`s`	The Jaccard similarity.

Details

Locality sensitive hashing returns a list of possible matches for similar documents. How likely is it that a pair of documents will be detected as a possible match? If `h` is the number of minhash signatures, `b` is the number of bands in the LSH function (implying then that the number of rows `r = h / b`), and `s` is the actual Jaccard similarity of the two documents, then the probability `p` that the two documents will be marked as a candidate pair is given by this equation.

$$p = 1 - (1 - s^r)^b$$

According to MMDS, that equation approximates an S-curve. This implies that there is a threshold (`t`) for `s` approximated by this equation.

$$t = \frac{1}{b}^{\frac{1}{r}}$$

References

Jure Leskovec, Anand Rajaraman, and Jeff Ullman, *Mining of Massive Datasets* (Cambridge University Press, 2011), ch. 3.

Examples

```
# Threshold for default values
lsh_threshold(h = 200, b = 40)

# Probability for varying values of s
lsh_probability(h = 200, b = 40, s = .25)
lsh_probability(h = 200, b = 40, s = .50)
lsh_probability(h = 200, b = 40, s = .75)
```

---

lsh_query — *Query a LSH cache for matches to a single document*

Description

This function retrieves the matches for a single document from an `lsh_buckets` object created by `lsh`. See `lsh_candidates` to retrieve all pairs of matches.

Usage

```
lsh_query(buckets, id)
```

Arguments

`buckets`	An `lsh_buckets` object created by `lsh`.
`id`	The document ID to find matches for.

Value

An `lsh_candidates` data frame with matches to the document specified.

See Also

`lsh`, `lsh_candidates`

Examples

```
dir <- system.file("extdata/legal", package = "textreuse")
minhash <- minhash_generator(200, seed = 235)
corpus <- TextReuseCorpus(dir = dir,
                          tokenizer = tokenize_ngrams, n = 5,
                          minhash_func = minhash)
buckets <- lsh(corpus, bands = 50)
lsh_query(buckets, "ny1850-match")
```

lsh_subset *List of all candidates in a corpus*

Description

List of all candidates in a corpus

Usage

```
lsh_subset(candidates)
```

Arguments

candidates	A data frame of candidate pairs from lsh_candidates.

Value

A character vector of document IDs from the candidate pairs, to be used to subset the TextReuseCorpus.

Examples

```
dir <- system.file("extdata/legal", package = "textreuse")
minhash <- minhash_generator(200, seed = 234)
corpus <- TextReuseCorpus(dir = dir,
                          tokenizer = tokenize_ngrams, n = 5,
                          minhash_func = minhash)
buckets <- lsh(corpus, bands = 50)
candidates <- lsh_candidates(buckets)
lsh_subset(candidates)
corpus[lsh_subset(candidates)]
```

minhash_generator *Generate a minhash function*

Description

A minhash value is calculated by hashing the strings in a character vector to integers and then selecting the minimum value. Repeated minhash values are generated by using different hash functions: these different hash functions are created by using performing a bitwise XOR operation (bitwXor) with a vector of random integers. Since it is vital that the same random integers be used for each document, this function generates another function which will always use the same integers. The returned function is intended to be passed to the hash_func parameter of TextReuseTextDocument.

Usage

```
minhash_generator(n = 200, seed = NULL)
```

Arguments

n	The number of minhashes that the returned function should generate.
seed	An option parameter to set the seed used in generating the random numbers to ensure that the same minhash function is used on repeated applications.

Value

A function which will take a character vector and return n minhashes.

References

Jure Leskovec, Anand Rajaraman, and Jeff Ullman, *Mining of Massive Datasets* (Cambridge University Press, 2011), ch. 3. See also Matthew Casperson, "Minhash for Dummies" (November 14, 2013).

See Also

lsh

Examples

```
set.seed(253)
minhash <- minhash_generator(10)

# Example with a TextReuseTextDocument
file <- system.file("extdata/legal/ny1850-match.txt", package = "textreuse")
doc <- TextReuseTextDocument(file = file, hash_func = minhash,
                             keep_tokens = TRUE)
hashes(doc)

# Example with a character vector
is.character(tokens(doc))
minhash(tokens(doc))
```

---

pairwise_candidates *Candidate pairs from pairwise comparisons*

Description

Converts a comparison matrix generated by pairwise_compare into a data frame of candidates for matches.

Usage

```
pairwise_candidates(m, directional = FALSE)
```

Arguments

m	A matrix from pairwise_compare.
directional	Should be set to the same value as in pairwise_compare.

Value

A data frame containing all the non-NA values from m. Columns a and b are the IDs from the original corpus as passed to the comparison function. Column score is the score returned by the comparison function.

Examples

```
dir <- system.file("extdata/legal", package = "textreuse")
corpus <- TextReuseCorpus(dir = dir)

m1 <- pairwise_compare(corpus, ratio_of_matches, directional = TRUE)
pairwise_candidates(m1, directional = TRUE)

m2 <- pairwise_compare(corpus, jaccard_similarity)
pairwise_candidates(m2)
```

---

pairwise_compare *Pairwise comparisons among documents in a corpus*

Description

Given a TextReuseCorpus containing documents of class TextReuseTextDocument, this function applies a comparison function to every pairing of documents, and returns a matrix with the comparison scores.

Usage

```
pairwise_compare(corpus, f, ..., directional = FALSE, progress = interactive())
```

Arguments

corpus	A TextReuseCorpus.
f	The function to apply to x and y.
...	Additional arguments passed to f.
directional	Some comparison functions are commutative, so that f(a, b) == f(b, a) (e.g., jaccard_similarity). Other functions are directional, so that f(a, b) measures a's borrowing from b, which may not be the same as f(b, a) (e.g., ratio_of_matches). If directional is FALSE, then only the minimum number of comparisons will be made, i.e., the upper triangle of the matrix. If directional is TRUE, then both directional comparisons will be measured. In no case, however, will documents be compared to themselves, i.e., the diagonal of the matrix.
progress	Display a progress bar while comparing documents.

Value

A square matrix with dimensions equal to the length of the corpus, and row and column names set by the names of the documents in the corpus. A value of NA in the matrix indicates that a comparison was not made. In cases of directional comparisons, then the comparison reported is f(row, column).

See Also

See these document comparison functions, jaccard_similarity, ratio_of_matches.

Examples

```
dir <- system.file("extdata/legal", package = "textreuse")
corpus <- TextReuseCorpus(dir = dir)
names(corpus) <- filenames(names(corpus))

# A non-directional comparison
pairwise_compare(corpus, jaccard_similarity)

# A directional comparison
pairwise_compare(corpus, ratio_of_matches, directional = TRUE)
```

---

rehash — *Recompute the hashes for a document or corpus*

Description

Given a TextReuseTextDocument or a TextReuseCorpus, this function recomputes either the hashes or the minhashes with the function specified. This implies that you have retained the tokens with the keep_tokens = TRUE parameter.

Usage

```
rehash(x, func, type = c("hashes", "minhashes"))
```

Arguments

x	A TextReuseTextDocument or TextReuseCorpus.
func	A function to either hash the tokens or to generate the minhash signature. See hash_string, minhash_generator.
type	Recompute the hashes or minhashes?

Value

The modified TextReuseTextDocument or TextReuseCorpus.

Examples

```
dir <- system.file("extdata/legal", package = "textreuse")
minhash1 <- minhash_generator(seed = 1)
corpus <- TextReuseCorpus(dir = dir, minhash_func = minhash1, keep_tokens = TRUE)
head(minhashes(corpus[[1]]))
minhash2 <- minhash_generator(seed = 2)
corpus <- rehash(corpus, minhash2, type = "minhashes")
head(minhashes(corpus[[2]]))
```

`similarity-functions` *Measure similarity/dissimilarity in documents*

Description

A set of functions which take two sets or bag of words and measure their similarity or dissimilarity.

Usage

```
jaccard_similarity(a, b)

jaccard_dissimilarity(a, b)

jaccard_bag_similarity(a, b)

ratio_of_matches(a, b)
```

Arguments

a	The first set (or bag) to be compared. The origin bag for directional comparisons.
b	The second set (or bag) to be compared. The destination bag for directional comparisons.

Details

The functions `jaccard_similarity` and `jaccard_dissimilarity` provide the Jaccard measures of similarity or dissimilarity for two sets. The coefficients will be numbers between `0` and `1`. For the similarity coefficient, the higher the number the more similar the two sets are. When applied to two documents of class `TextReuseTextDocument`, the hashes in those documents are compared. But this function can be passed objects of any class accepted by the set functions in base R. So it is possible, for instance, to pass this function two character vectors comprised of word, line, sentence, or paragraph tokens, or those character vectors hashed as integers.

The Jaccard similarity coeffecient is defined as follows:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

The Jaccard dissimilarity is simply

$$1 - J(A, B)$$

The function `jaccard_bag_similarity` treats `a` and `b` as bags rather than sets, so that the result is a fraction where the numerator is the sum of each matching element counted the minimum number of times it appears in each bag, and the denominator is the sum of the lengths of both bags. The maximum value for the Jaccard bag similarity is `0.5`.

The function `ratio_of_matches` finds the ratio between the number of items in `b` that are also in `a` and the total number of items in `b`. Note that this similarity measure is directional: it measures how much `b` borrows from `a`, but says nothing about how much of `a` borrows from `b`.

References

Jure Leskovec, Anand Rajaraman, and Jeff Ullman, *Mining of Massive Datasets* (Cambridge University Press, 2011).

Examples

```
jaccard_similarity(1:6, 3:10)
jaccard_dissimilarity(1:6, 3:10)

a <- c("a", "a", "a", "b")
b <- c("a", "a", "b", "b", "c")
jaccard_similarity(a, b)
jaccard_bag_similarity(a, b)
ratio_of_matches(a, b)
ratio_of_matches(b, a)

ny         <- system.file("extdata/legal/ny1850-match.txt", package = "textreuse")
ca_match   <- system.file("extdata/legal/ca1851-match.txt", package = "textreuse")
ca_nomatch <- system.file("extdata/legal/ca1851-nomatch.txt", package = "textreuse")

ny         <- TextReuseTextDocument(file = ny,
                                    meta = list(id = "ny"))
ca_match   <- TextReuseTextDocument(file = ca_match,
                                    meta = list(id = "ca_match"))
ca_nomatch <- TextReuseTextDocument(file = ca_nomatch,
                                    meta = list(id = "ca_nomatch"))

# These two should have higher similarity scores
jaccard_similarity(ny, ca_match)
ratio_of_matches(ny, ca_match)

# These two should have lower similarity scores
jaccard_similarity(ny, ca_nomatch)
ratio_of_matches(ny, ca_nomatch)
```

TextReuseCorpus *TextReuseCorpus*

Description

This is the constructor function for a TextReuseCorpus, modeled on the virtual S3 class Corpus from the tm package. The object is a TextReuseCorpus, which is basically a list containing objects of class TextReuseTextDocument. Arguments are passed along to that constructor function. To create the corpus, you can pass either a character vector of paths to text files using the paths = parameter, a directory containing text files (with any extension) using the dir = parameter, or a character vector of documents using the text = parameter, where each element in the characer vector is a document. If the character vector passed to text = has names, then those names will be used as the document IDs. Otherwise, IDs will be assigned to the documents. Only one of the paths, dir, or text parameters should be specified.

Usage

```
TextReuseCorpus(
  paths,
  dir = NULL,
  text = NULL,
  meta = list(),
  progress = interactive(),
  tokenizer = tokenize_ngrams,
  ...,
  hash_func = hash_string,
  minhash_func = NULL,
  keep_tokens = FALSE,
  keep_text = TRUE,
  skip_short = TRUE
)

is.TextReuseCorpus(x)

skipped(x)
```

Arguments

paths	A character vector of paths to files to be opened.
dir	The path to a directory of text files.
text	A character vector (possibly named) of documents.
meta	A list with named elements for the metadata associated with this corpus.
progress	Display a progress bar while loading files.
tokenizer	A function to split the text into tokens. See tokenizers. If value is NULL, then tokenizing and hashing will be skipped.

...	Arguments passed on to the tokenizer.
hash_func	A function to hash the tokens. See hash_string.
minhash_func	A function to create minhash signatures of the document. See minhash_generator.
keep_tokens	Should the tokens be saved in the documents that are returned or discarded?
keep_text	Should the text be saved in the documents that are returned or discarded?
skip_short	Should short documents be skipped? (See details.)
x	An R object to check.

Details

If skip_short = TRUE, this function will skip very short or empty documents. A very short document is one where there are two few words to create at least two n-grams. For example, if five-grams are desired, then a document must be at least six words long. If no value of n is provided, then the function assumes a value of n = 3. A warning will be printed with the document ID of each skipped document. Use skipped() to get the IDs of skipped documents.

This function will use multiple cores on non-Windows machines if the "mc.cores" option is set. For example, to use four cores: options("mc.cores" = 4L).

See Also

Accessors for TextReuse objects.

Examples

```
dir <- system.file("extdata/legal", package = "textreuse")
corpus <- TextReuseCorpus(dir = dir, meta = list("description" = "Field Codes"))
# Subset by position or file name
corpus[[1]]
names(corpus)
corpus[["ca1851-match"]]
```

TextReuseTextDocument *TextReuseTextDocument*

Description

This is the constructor function for TextReuseTextDocument objects. This class is used for comparing documents.

Usage

```
TextReuseTextDocument(
  text,
  file = NULL,
  meta = list(),
  tokenizer = tokenize_ngrams,
  ...,
  hash_func = hash_string,
  minhash_func = NULL,
  keep_tokens = FALSE,
  keep_text = TRUE,
  skip_short = TRUE
)

is.TextReuseTextDocument(x)

has_content(x)

has_tokens(x)

has_hashes(x)

has_minhashes(x)
```

Arguments

text	A character vector containing the text of the document. This argument can be skipped if supplying `file`.
file	The path to a text file, if `text` is not provided.
meta	A list with named elements for the metadata associated with this document. If a document is created using the `text` parameter, then you must provide an `id` field, e.g., `meta = list(id = "my_id")`. If the document is created using `file`, then the ID will be created from the file name.
tokenizer	A function to split the text into tokens. See `tokenizers`. If value is `NULL`, then tokenizing and hashing will be skipped.
...	Arguments passed on to the `tokenizer`.
hash_func	A function to hash the tokens. See `hash_string`.
minhash_func	A function to create minhash signatures of the document. See `minhash_generator`.
keep_tokens	Should the tokens be saved in the document that is returned or discarded?
keep_text	Should the text be saved in the document that is returned or discarded?
skip_short	Should short documents be skipped? (See details.)
x	An R object to check.

Details

This constructor function follows a three-step process. It reads in the text, either from a file or from memory. It then tokenizes that text. Then it hashes the tokens. Most of the comparison functions in this package rely only on the hashes to make the comparison. By passing `FALSE` to `keep_tokens` and `keep_text`, you can avoid saving those objects, which can result in significant memory savings for large corpora.

If `skip_short = TRUE`, this function will return `NULL` for very short or empty documents. A very short document is one where there are two few words to create at least two n-grams. For example, if five-grams are desired, then a document must be at least six words long. If no value of `n` is provided, then the function assumes a value of `n = 3`. A warning will be printed with the document ID of a skipped document.

Value

An object of class `TextReuseTextDocument`. This object inherits from the virtual S3 class `TextDocument` in the NLP package. It contains the following elements:

content The text of the document.

tokens The tokens created from the text.

hashes Hashes created from the tokens.

minhashes The minhash signature of the document.

metadata The document metadata, including the filename (if any) in `file`.

See Also

Accessors for TextReuse objects.

Examples

```
file <- system.file("extdata/legal/ny1850-match.txt", package = "textreuse")
doc  <- TextReuseTextDocument(file = file, meta = list(id = "ny1850"))
print(doc)
meta(doc)
head(tokens(doc))
head(hashes(doc))
## Not run:
content(doc)

## End(Not run)
```

TextReuseTextDocument-accessors

Accessors for TextReuse objects

Description

Accessor functions to read and write components of `TextReuseTextDocument` and `TextReuseCorpus` objects.

Usage

```
tokens(x)

tokens(x) <- value

hashes(x)

hashes(x) <- value

minhashes(x)

minhashes(x) <- value
```

Arguments

`x`	The object to access.
`value`	The value to assign.

Value

Either a vector or a named list of vectors.

tokenize

Recompute the tokens for a document or corpus

Description

Given a `TextReuseTextDocument` or a `TextReuseCorpus`, this function recomputes the tokens and hashes with the functions specified. Optionally, it can also recompute the minhash signatures.

Usage

```
tokenize(
  x,
  tokenizer,
  ...,
  hash_func = hash_string,
  minhash_func = NULL,
  keep_tokens = FALSE,
  keep_text = TRUE
)
```

Arguments

x	A TextReuseTextDocument or TextReuseCorpus.
tokenizer	A function to split the text into tokens. See tokenizers.
...	Arguments passed on to the tokenizer.
hash_func	A function to hash the tokens. See hash_string.
minhash_func	A function to create minhash signatures. See minhash_generator.
keep_tokens	Should the tokens be saved in the document that is returned or discarded?
keep_text	Should the text be saved in the document that is returned or discarded?

Value

The modified TextReuseTextDocument or TextReuseCorpus.

Examples

```
dir <- system.file("extdata/legal", package = "textreuse")
corpus <- TextReuseCorpus(dir = dir, tokenizer = NULL)
corpus <- tokenize(corpus, tokenize_ngrams)
head(tokens(corpus[[1]]))
```

---

tokenizers *Split texts into tokens*

Description

These functions each turn a text into tokens. The tokenize_ngrams functions returns shingled n-grams.

Usage

```
tokenize_words(string, lowercase = TRUE)

tokenize_sentences(string, lowercase = TRUE)

tokenize_ngrams(string, lowercase = TRUE, n = 3)

tokenize_skip_ngrams(string, lowercase = TRUE, n = 3, k = 1)
```

Arguments

`string`	A character vector of length 1 to be tokenized.
`lowercase`	Should the tokens be made lower case?
`n`	For n-gram tokenizers, the number of words in each n-gram.
`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

These functions will strip all punctuation.

Value

A character vector containing the tokens.

Examples

```
dylan <- "How many roads must a man walk down? The answer is blowin' in the wind."
tokenize_words(dylan)
tokenize_sentences(dylan)
tokenize_ngrams(dylan, n = 2)
tokenize_skip_ngrams(dylan, n = 3, k = 2)
```

`wordcount` *Count words*

Description

This function counts words in a text, for example, a character vector, a `TextReuseTextDocument`, some other object that inherits from `TextDocument`, or a all the documents in a `TextReuseCorpus`.

Usage

```
wordcount(x)
```

Arguments

`x`	The object containing a text.

Value

An integer vector for the word count.

Index