Package ‘tm’

November 18, 2017

Title Text Mining Package
Version 0.7-2
Date 2017-11-17
Depends R (>= 3.2.0), NLP (>= 0.1-6.2)
Imports Rcpp, parallel, slam (>= 0.1-37), stats, tools, utils, graphics, xml2
LinkingTo BH, Rcpp
Suggests antiword, filehash, methods, pdftools, Rcmapdf, Rgraphviz, Rpoppler, SnowballC, testthat, tm.lexicon.GeneralInquirer
SystemRequirements C++11
Description A framework for text mining applications within R.
License GPL-3
URL http://tm.r-forge.r-project.org/
Additional_repositories http://datacube.wu.ac.at
NeedsCompilation yes
Author Ingo Feinerer [aut, cre],
Kurt Hornik [aut] <https://orcid.org/0000-0003-4198-9911>,
Artifex Software, Inc. [ctb, cph] (pdf_info.ps taken from GPL Ghostscript)
Maintainer Ingo Feinerer <feinerer@logic.at>
Repository CRAN
Date/Publication 2017-11-18 17:23:28 UTC

R topics documented:

acq .......................................................... 3
corpus_transformer ........................................ 4
Corpus ...................................................... 4
crude ....................................................... 5
DataframeSource ......................................... 6
<table>
<thead>
<tr>
<th>R topics documented:</th>
</tr>
</thead>
<tbody>
<tr>
<td>DirSource</td>
</tr>
<tr>
<td>Docs</td>
</tr>
<tr>
<td>findAssocs</td>
</tr>
<tr>
<td>findFreqTerms</td>
</tr>
<tr>
<td>findMostFreqTerms</td>
</tr>
<tr>
<td>foreign</td>
</tr>
<tr>
<td>getTokenizers</td>
</tr>
<tr>
<td>getTransformations</td>
</tr>
<tr>
<td>hpc</td>
</tr>
<tr>
<td>inspect</td>
</tr>
<tr>
<td>meta</td>
</tr>
<tr>
<td>PCorpus</td>
</tr>
<tr>
<td>PlainTextDocument</td>
</tr>
<tr>
<td>plot</td>
</tr>
<tr>
<td>readDataframe</td>
</tr>
<tr>
<td>readDOC</td>
</tr>
<tr>
<td>Reader</td>
</tr>
<tr>
<td>readPDF</td>
</tr>
<tr>
<td>readPlain</td>
</tr>
<tr>
<td>readRCV1</td>
</tr>
<tr>
<td>readReut21578XML</td>
</tr>
<tr>
<td>readTagged</td>
</tr>
<tr>
<td>readXML</td>
</tr>
<tr>
<td>removeNumbers</td>
</tr>
<tr>
<td>removePunctuation</td>
</tr>
<tr>
<td>removeSparseTerms</td>
</tr>
<tr>
<td>removeWords</td>
</tr>
<tr>
<td>SimpleCorpus</td>
</tr>
<tr>
<td>Source</td>
</tr>
<tr>
<td>stemCompletion</td>
</tr>
<tr>
<td>stemDocument</td>
</tr>
<tr>
<td>stopwords</td>
</tr>
<tr>
<td>stripWhitespace</td>
</tr>
<tr>
<td>TermDocumentMatrix</td>
</tr>
<tr>
<td>termFreq</td>
</tr>
<tr>
<td>TextDocument</td>
</tr>
<tr>
<td>tm_combine</td>
</tr>
<tr>
<td>tm_filter</td>
</tr>
<tr>
<td>tm_map</td>
</tr>
<tr>
<td>tm_reduce</td>
</tr>
<tr>
<td>tm_term_score</td>
</tr>
<tr>
<td>tokenizer</td>
</tr>
<tr>
<td>URIsource</td>
</tr>
<tr>
<td>VCorpus</td>
</tr>
<tr>
<td>VectorSource</td>
</tr>
<tr>
<td>weightBin</td>
</tr>
<tr>
<td>WeightFunction</td>
</tr>
<tr>
<td>weightSMART</td>
</tr>
</tbody>
</table>
Description

This dataset holds 50 news articles with additional meta information from the Reuters-21578 data set. All documents belong to the topic `acq` dealing with corporate acquisitions.

Usage

data("acq")

Format

A `vCorpus` of 50 text documents.

Source

Reuters-21578 Text Categorization Collection Distribution 1.0 (XML format).

References


Luz, Saturnino *XML-encoded version of Reuters-21578*. https://www.scss.tcd.ie/~luzs/t/cs4114/sw/reuters21578~xml/

Examples

data("acq")
acq
content_transformer  Content Transformers

Description
Create content transformers, i.e., functions which modify the content of an R object.

Usage
content_transformer(FUN)

Arguments
FUN  a function.

Value
A function with two arguments:
  x  an R object with implemented content getter (content) and setter (content<-) functions.
  ... arguments passed over to FUN.

See Also
tm_map for an interface to apply transformations to corpora.

Examples
data("crude")
crude[[1]]
(f <- content_transformer(function(x, pattern) gsub(pattern, ",", x)))
tm_map(crude, f, "[[:digit:]]*")[[1]]

Corpus  Corpora

Description
Representing and computing on corpora.
Details

Corpora are collections of documents containing (natural language) text. In packages which employ the infrastructure provided by package `tm`, such corpora are represented via the virtual S3 class `Corpus`: such packages then provide S3 corpus classes extending the virtual base class (such as `VCorpus` provided by package `tm` itself).

All extension classes must provide accessors to extract subsets ([]), individual documents ([[]), and metadata (`meta`). The function `length` must return the number of documents, and `as.list` must construct a list holding the documents.

A corpus can have two types of metadata (accessible via `meta`). Corpus metadata contains corpus specific metadata in form of tag-value pairs. Document level metadata contains document specific metadata but is stored in the corpus as a data frame. Document level metadata is typically used for semantic reasons (e.g., classifications of documents form an own entity due to some high-level information like the range of possible values) or for performance reasons (single access instead of extracting metadata of each document).

The function `Corpus` is a convenience alias to `SimpleCorpus` or `VCorpus`, depending on the arguments provided.

See Also

`SimpleCorpus`, `VCorpus`, and `PCorpus` for the corpora classes provided by package `tm`.

`DCorpus` for a distributed corpus class provided by package `tm.plugin.dc`.

Description

This data set holds 20 news articles with additional meta information from the Reuters-21578 data set. All documents belong to the topic `crude` dealing with crude oil.

Usage

data("crude")

Format

A `VCorpus` of 20 text documents.

Source

Reuters-21578 Text Categorization Collection Distribution 1.0 (XML format).
References


Examples

data("crude")
crude

DataframeSource Data Frame Source

Description

Create a data frame source.

Usage

DataframeSource(x)

Arguments

x A data frame giving the texts and metadata.

Details

A data frame source interprets each row of the data frame x as a document. The first column must be named "doc_id" and contain a unique string identifier for each document. The second column must be named "text" and contain a "UTF-8" encoded string representing the document's content. Optional additional columns are used as document level metadata.

Value

An object inheriting from DataframeSource, SimpleSource, and Source.

See Also

Source for basic information on the source infrastructure employed by package tm, and meta for types of metadata. readtext for reading in a text in multiple formats suitable to be processed by DataframeSource.
DirSource

Examples

docs <- data.frame(doc_id = c("doc_1", "doc_2"),
                   text = c("This is a text.", "This another one."),
                   dmeta1 = 1:2, dmeta2 = letters[1:2],
                   stringsAsFactors = FALSE)
(ds <- DataFrameSource(docs))
x <- Corpus(ds)
inspect(x)
meta(x)

DirSource  Directory Source

Description

Create a directory source.

Usage

DirSource(directory = ",", encoding = ",",
          pattern = NULL,
          recursive = FALSE,
          ignore.case = FALSE,
          mode = "text")

Arguments

directory A character vector of full path names; the default corresponds to the working
directory getwd().
encoding a character string describing the current encoding. It is passed to iconv to con-
vert the input to UTF-8.
pattern an optional regular expression. Only file names which match the regular expres-
sion will be returned.
recursive logical. Should the listing recurse into directories?
ignore.case logical. Should pattern-matching be case-insensitive?
mode a character string specifying if and how files should be read in. Available modes are:
        "" No read. In this case getElem and pGetElem only deliver URIs.
        "binary" Files are read in binary raw mode (via readBin).
        "text" Files are read as text (via readLines).

Details

A directory source acquires a list of files via dir and interprets each file as a document.
Value
An object inheriting from DirSource, SimpleSource, and Source.

See Also
Source for basic information on the source infrastructure employed by package tm. Encoding and iconv on encodings.

Examples
DirSource(system.file("texts", "txt", package = "tm"))

docs Access Document IDs and Terms

Description
Accessing document IDs, terms, and their number of a term-document matrix or document-term matrix.

Usage
Docs(x)
nDocs(x)
nTerms(x)
Terms(x)

Arguments
x Either a TermDocumentMatrix or DocumentTermMatrix.

Value
For Docs and Terms, a character vector with document IDs and terms, respectively.
For nDocs and nTerms, an integer with the number of document IDs and terms, respectively.

Examples
data("crude")
tdm <- TermDocumentMatrix(crude)[1:10,1:20]
Docs(tdm)
nDocs(tdm)
nTerms(tdm)
Terms(tdm)
findAssocs

Find Associations in a Term-Document Matrix

Description

Find associations in a document-term or term-document matrix.

Usage

```r
## S3 method for class 'DocumentTermMatrix'
findAssocs(x, terms, corlimit)
## S3 method for class 'TermDocumentMatrix'
findAssocs(x, terms, corlimit)
```

Arguments

- `x` A `DocumentTermMatrix` or a `TermDocumentMatrix`.
- `terms` a character vector holding terms.
- `corlimit` a numeric vector (of the same length as `terms`; recycled otherwise) for the (inclusive) lower correlation limits of each term in the range from zero to one.

Value

A named list. Each list component is named after a term in `terms` and contains a named numeric vector. Each vector holds matching terms from `x` and their rounded correlations satisfying the inclusive lower correlation limit of `corlimit`.

Examples

```r
data("crude")
tdm <- TermDocumentMatrix(crude)
findAssocs(tdm, c("oil", "opec", "xyz"), c(0.7, 0.75, 0.1))
```

findFreqTerms

Find Frequent Terms

Description

Find frequent terms in a document-term or term-document matrix.

Usage

```r
findFreqTerms(x, lowfreq = 0, highfreq = Inf)
```
Arguments

x A DocumentTermMatrix or TermDocumentMatrix.

lowfreq A numeric for the lower frequency bound.

highfreq A numeric for the upper frequency bound.

Details

This method works for all numeric weightings but is probably most meaningful for the standard term frequency (tf) weighting of x.

Value

A character vector of terms in x which occur more or equal often than lowfreq times and less or equal often than highfreq times.

Examples

data("crude")
tdm <- TermDocumentMatrix(crude)
findFreqTerms(tdm, 2, 3)

findMostFreqTerms Find Most Frequent Terms

Description

Find most frequent terms in a document-term or term-document matrix, or a vector of term frequencies.

Usage

findMostFreqTerms(x, n = 6L, ...)
## S3 method for class 'DocumentTermMatrix'
findMostFreqTerms(x, n = 6L, INDEX = NULL, ...)
## S3 method for class 'TermDocumentMatrix'
findMostFreqTerms(x, n = 6L, INDEX = NULL, ...)

Arguments

x A DocumentTermMatrix or TermDocumentMatrix, or a vector of term frequencies as obtained by termFreq().

n A single integer giving the maximal number of terms.

INDEX an object specifying a grouping of documents for rollup, or NULL (default) in which case each document is considered individually.

... arguments to be passed to or from methods.
Details

Only terms with positive frequencies are included in the results.

Value

For the document-term or term-document matrix methods, a list with the named frequencies of the up to \( n \) most frequent terms occurring in each document (group). Otherwise, a single such vector of most frequent terms.

Examples

```r
data("crude")

## Term frequencies:
tf <- termFreq(crude[[1L]])
findMostFreqTerms(tf)

## Document-term matrices:
dtm <- DocumentTermMatrix(crude)
## Most frequent terms for each document:
findMostFreqTerms(dtm)
## Most frequent terms for the first 10 the second 10 documents, respectively:
findMostFreqTerms(dtm, INDEX = rep(1:2, each = 10L))
```

---

**foreign**

Read Document-Term Matrices

**Description**

Read document-term matrices stored in special file formats.

**Usage**

```r
read_dtm_Blei_et_al(file, vocab = NULL)
read_dtm_MC(file, scalingtype = NULL)
```

**Arguments**

- `file` a character string with the name of the file to read.
- `vocab` a character string with the name of a vocabulary file (giving the terms, one per line), or NULL.
- `scalingtype` a character string specifying the type of scaling to be used, or NULL (default), in which case the scaling will be inferred from the names of the files with non-zero entries found (see Details).
Details


MC is a toolkit for creating vector models from text documents (see http://www.cs.utexas.edu/users/dml/software/mc/). It employs a variant of Compressed Column Storage (CCS) sparse matrix format, writing data into several files with suitable names: e.g., a file with `_dim` appended to the base file name stores the matrix dimensions. The non-zero entries are stored in a file the name of which indicates the scaling type used: e.g., `_tfx_nz` indicates scaling by term frequency (`t`), inverse document frequency (`f`) and no normalization (`x`). See ‘README’ in the MC sources for more information.

`read_dtm_MC` reads such sparse matrix information with argument file giving the path with the base file name.

Value

A document-term matrix.

See Also

`read_stm_MC` in package `slam`.

---

### getTokenizers

**Description**

Predefined tokenizers.

**Usage**

getTokenizers()

**Value**

A character vector with tokenizers provided by package `tm`.

**See Also**

`Boost_tokenizer`, `MC_tokenizer` and `scan_tokenizer`.

**Examples**

getTokenizers()
**getTransformations**

**Transformations**

**Description**
Predefined transformations (mappings) which can be used with `tm_map`.

**Usage**
`getTransformations()`

**Value**
A character vector with transformations provided by package `tm`.

**See Also**
`removeNumbers`, `removePunctuation`, `removeWords`, `stemDocument`, and `stripWhitespace`, `content_transformer` to create custom transformations.

**Examples**
`getTransformations()`

---

**hpc**

**Parallelized `lapply`**

**Description**
Parallelize applying a function over a list or vector according to the registered parallelization engine.

**Usage**
`tm_parLapply(X, FUN, ...)`
`tm_parLapply_engine(new)`

**Arguments**

- **X**
  A vector (atomic or list), or other objects suitable for the engine in use.

- **FUN**
  the function to be applied to each element of `X`.

- **...**
  optional arguments to `FUN`.

- **new**
  an object inheriting from class `cluster` as created by `makeCluster()` from package `parallel`, or a function with formals `X`, `FUN` and `...`, or NULL corresponding to the default of using no parallelization engine.
Details

Parallelization can be employed to speed up some of the embarrassingly parallel computations performed in package \texttt{tm}, specifically \texttt{tm_index()}, \texttt{tm_map()} on a non-lazy-mapped \texttt{VCorpus}, and \texttt{TermDocumentMatrix()} on a \texttt{VCorpus} or \texttt{PCorpus}. Functions \texttt{tm_parLapply()} and \texttt{tm_parLapply_engine()} can be used to customize parallelization according to the available resources.

\texttt{tm_parLapply_engine()} is used for getting (with no arguments) or setting (with argument \texttt{new}) the parallelization engine employed (see below for examples).

If an engine is set to an object inheriting from class \texttt{cluster}, \texttt{tm_parLapply()} calls \texttt{parLapply()} with this cluster and the given arguments. If set to a function, \texttt{tm_parLapply()} calls the function with the given arguments. Otherwise, it simply calls \texttt{lapply()}.

Hence, to achieve parallelization via \texttt{parLapply()} and a default cluster registered via \texttt{setDefaultCluster()}, one can use

\begin{verbatim}
  tm_parLapply_engine(function(X, FUN, ...)
    parallel::parLapply(NULL, X, FUN, ...))
\end{verbatim}

or re-register the cluster, say \texttt{cl}, using

\begin{verbatim}
  tm_parLapply_engine(cl)
\end{verbatim}

(note that there is no mechanism for programmatically getting the registered default cluster). Using

\begin{verbatim}
  tm_parLapply_engine(function(X, FUN, ...)
    parallel::parLapplyLB(NULL, X, FUN, ...))
\end{verbatim}

or

\begin{verbatim}
  tm_parLapply_engine(function(X, FUN, ...)
    parallel::parLapplyLB(cl, X, FUN, ...))
\end{verbatim}

gives load-balancing parallelization with the registered default or given cluster, respectively. To achieve parallelization via forking (on Unix-alike platforms), one can use the above with clusters created by \texttt{makeForkCluster()}, or use

\begin{verbatim}
  tm_parLapply_engine(parallel::mclapply)
\end{verbatim}

or

\begin{verbatim}
  tm_parLapply_engine(function(X, FUN, ...)
    parallel::mclapply(X, FUN, ..., mc.cores = n))
\end{verbatim}

to use \texttt{mclapply()} with the default or given number \texttt{n} of cores.

Value

A list the length of \texttt{X}, with the result of applying \texttt{FUN} together with the \ldots arguments to each element of \texttt{X}.

See Also

\texttt{makeCluster()}, \texttt{parLapply()}, \texttt{parLapplyLB()}, and \texttt{mclapply()}.  


inspect

---

**Inspect Objects**

**Description**

Inspect, i.e., display detailed information on a corpus, a term-document matrix, or a text document.

**Usage**

```
## S3 method for class 'PCorpus'
inspect(x)
## S3 method for class 'VCorpus'
inspect(x)
## S3 method for class 'TermDocumentMatrix'
inspect(x)
## S3 method for class 'TextDocument'
inspect(x)
```

**Arguments**

- `x` Either a corpus, a term-document matrix, or a text document.

**Examples**

```
data("crude")
inspect(crude[1:3])
inspect(crude[[1]])
tdm <- TermDocumentMatrix(crude)[1:10, 1:10]
inspect(tdm)
```

---

**meta**

---

**Metadata Management**

**Description**

Accessing and modifying metadata of text documents and corpora.

**Usage**

```
## S3 method for class 'PCorpus'
meta(x, tag = NULL, type = c("indexed", "corpus", "local"), ...)
## S3 replacement method for class 'PCorpus'
meta(x, tag, type = c("indexed", "corpus", "local"), ...) <- value
## S3 method for class 'SimpleCorpus'
meta(x, tag = NULL, type = c("indexed", "corpus"), ...)
## S3 replacement method for class 'SimpleCorpus'
```

Arguments

- `x` For `dublincore` a `TextDocument`, and for `meta` a `TextDocument` or a `Corpus`.
- `tag` a character giving the name of a metadatum. No tag corresponds to all available metadata.
- `type` a character specifying the kind of corpus metadata (see Details).
- `value` replacement value.

Details

A corpus has two types of metadata. *Corpus metadata* ("corpus") contains corpus specific metadata in form of tag-value pairs. *Document level metadata* ("indexed") contains document specific metadata but is stored in the corpus as a data frame. Document level metadata is typically used for semantic reasons (e.g., classifications of documents form an own entity due to some high-level information like the range of possible values) or for performance reasons (single access instead of extracting metadata of each document). The latter can be seen as a from of indexing, hence the name "indexed". *Document metadata* ("local") are tag-value pairs directly stored locally at the individual documents.

`dublincore` is a convenience wrapper to access and modify the metadata of a text document using the Simple Dublin Core schema (supporting the 15 metadata elements from the Dublin Core Metadata Element Set [http://dublincore.org/documents/dces/](http://dublincore.org/documents/dces/)).

References


See Also

`meta` for metadata in package `NLP`. 

```r
meta(x, tag, type = c("indexed", "corpus"), ...) <- value
## S3 method for class 'VCorpus'
meta(x, tag = NULL, type = c("indexed", "corpus", "local"), ...) <- value
## S3 replacement method for class 'VCorpus'
meta(x, tag, type = c("indexed", "corpus", "local"), ...) <- value
## S3 method for class 'PlainTextDocument'
meta(x, tag = NULL, ...)
## S3 replacement method for class 'PlainTextDocument'
meta(x, tag = NULL, ...) <- value
## S3 method for class 'XMLTextDocument'
meta(x, tag = NULL, ...)
## S3 replacement method for class 'XMLTextDocument'
meta(x, tag = NULL, ...) <- value
DublinCore(x, tag = NULL)
DublinCore(x, tag) <- value
```
Examples

```r
data("crude")
meta(crude[[1]])
DublinCore(crude[[1]])
meta(crude[[1]], tag = "topics")
meta(crude[[1]], tag = "comment") <- "A short comment."
meta(crude[[1]], tag = "topics") <- NULL
DublinCore(crude[[1]], tag = "creator") <- "Anonymous"
DublinCore(crude[[1]], tag = "format") <- "XML"
DublinCore(crude[[1]])
meta(crude[[1]])
meta(crude)
meta(crude, type = "corpus")
meta(crude, "labels") <- 21:40
meta(crude)
```

---

**PCorpus**

**Permanent Corpora**

**Description**

Create permanent corpora.

**Usage**

```r
PCorpus(x,  
  readerControl = list(reader = reader(x), language = "en"),  
  dbControl = list(dbName = "", dbType = "DB1"))
```

**Arguments**

- **x** A `Source` object.
- **readerControl** a named list of control parameters for reading in content from x.
  - **reader** a function capable of reading in and processing the format delivered by x.
  - **language** a character giving the language (preferably as IETF language tags, see `language` in package `NLP`). The default language is assumed to be English ("en").
- **dbControl** a named list of control parameters for the underlying database storage provided by package `filehash`.
  - **dbName** a character giving the filename for the database.
  - **dbType** a character giving the database format (see `filehashOption` for possible database formats).
Details

A permanent corpus stores documents outside of R in a database. Since multiple PCorpus R objects with the same underlying database can exist simultaneously in memory, changes in one get propagated to all corresponding objects (in contrast to the default R semantics).

Value

An object inheriting from PCorpus and Corpus.

See Also

Corpus for basic information on the corpus infrastructure employed by package tm.

VCorpus provides an implementation with volatile storage semantics.

Examples

txt <- system.file("texts", "txt", package = "tm")
## Not run: PCorpus(DirSource(txt),
    dbControl = list(dbName = "pcorpus.db", dbType = "DBI"))
## End(Not run)

Description

Create plain text documents.

Usage

PlainTextDocument(x = character(0),
    author = character(0),
    datetimestamp = as.POSIXlt(Sys.time(), tz = "GMT"),
    description = character(0),
    heading = character(0),
    id = character(0),
    language = character(0),
    origin = character(0),
    ...,
    meta = NULL,
    class = NULL)
Arguments

- `x`: A character giving the plain text content.
- `author`: A character or an object of class `person` giving the author names.
- `datetimestamp`: An object of class `POSIXt` or a character string giving the creation date/time information. If a character string, exactly one of the ISO 8601 formats defined by `http://www.w3.org/TR/NOTE-datetime` should be used. See `parse_ISO_8601_datetime` in package `NLP` for processing such date/time information.
- `description`: A character giving a description.
- `heading`: A character giving the title or a short heading.
- `id`: A character giving a unique identifier.
- `language`: A character giving the language (preferably as IETF language tags, see `language` in package `NLP`).
- `origin`: A character giving information on the source and origin.
- `...`: User-defined document metadata tag-value pairs.
- `meta`: A named list or NULL (default) giving all metadata. If set all other metadata arguments are ignored.
- `class`: A character vector or NULL (default) giving additional classes to be used for the created plain text document.

Value

An object inheriting from `class`, `PlainTextDocument` and `TextDocument`.

See Also

`TextDocument` for basic information on the text document infrastructure employed by package `tm`.

Examples

```r
(ptd <- PlainTextDocument("A simple plain text document",
                           heading = "Plain text document",
                           id = basename(tempfile()),
                           language = "en")

meta(ptd)
```

plot

Visualize a Term-Document Matrix

Description

Visualize correlations between terms of a term-document matrix.
Usage

```r
## S3 method for class 'TermDocumentMatrix'
plot(x, 
    terms = sample(Terms(x), 20), 
    corThreshold = 0.7, 
    weighting = FALSE, 
    attrs = list(graph = list(rankdir = "BT"), 
                 node = list(shape = "rectangle", 
                             fixedsize = FALSE)), 
    ...)```

Arguments

- **x**: A term-document matrix.
- **terms**: Terms to be plotted. Defaults to 20 randomly chosen terms of the term-document matrix.
- **corThreshold**: Do not plot correlations below this threshold. Defaults to 0.7.
- **weighting**: Define whether the line width corresponds to the correlation.
- **attrs**: Argument passed to the plot method for class `graphNEL`.
- **...**: Other arguments passed to the `graphNEL` plot method.

Details

Visualization requires that package `Rgraphviz` is available.

Examples

```r
## Not run: data(crude)
library(tidytext)
library(dplyr)
library(ggplot2)
library(Rgraphviz)

tdm <- TermDocumentMatrix(crude, 
                           control = list(removePunctuation = TRUE, 
                                          removeNumbers = TRUE, 
                                          stopwords = TRUE))

plot(tdm, corThreshold = 0.2, weighting = TRUE)
## End(Not run)
```

readDataframe  

---

**Description**

Read in a text document from a row in a data frame.

**Usage**

```r
readDataframe(elem, language, id)```
Arguments

- **elem**: A named list with the component `content` which must hold a data frame with rows as the documents to be read in. The names of the columns holding the text content and the document identifier must be "text" and "doc_id", respectively.
- **language**: A string giving the language.
- **id**: Not used.

Value

A `PlainTextDocument` representing `elem$content`.

See Also

- `Reader` for basic information on the reader infrastructure employed by package `tm`.

Examples

```r
docs <- data.frame(doc_id = c("doc_1", "doc_2"),
                   text = c("This is a text.", "This another one."),
                   stringsAsFactors = FALSE)
ds <- DataframeSource(docs)
elem <- getElem(stepNext(ds))
result <- readDataframe(elem, "en", NULL)
inspect(result)
meta(result)
```

---

**readDOC**

Read In a MS Word Document

Description

Return a function which reads in a Microsoft Word document extracting its text.

Usage

```r
readDOC(engine = c("antiword", "executable"), AntiwordOptions = "")
```

Arguments

- **engine**: A character string for the preferred DOC extraction engine (see Details).
- **AntiwordOptions**: Options passed over to antiword executable.
Details

Formally this function is a function generator, i.e., it returns a function (which reads in a text document) with a well-defined signature, but can access passed over arguments (e.g., options to antiword) via lexical scoping.

Available DOC extraction engines are as follows.

“antiword” (default) Antiword utility as provided by the function antiword in package antiword.
“executable” command line antiword executable which must be installed and accessible on your system. This can convert documents from Microsoft Word version 2, 6, 7, 97, 2000, 2002 and 2003 to plain text, and is available from http://www.winfield.demon.nl/. The character vector AntiwordOptions is passed over to the executable.

Value

A function with the following formals:

- `elem` a list with the named component `uri` which must hold a valid file name.
- `language` a string giving the language.
- `id` Not used.

The function returns a `PlainTextDocument` representing the text and metadata extracted from `elem$uri`.

See Also

Reader for basic information on the reader infrastructure employed by package tm.

Reader	Readers

Description

Creating readers.

Usage

getReaders()

Details

Readers are functions for extracting textual content and metadata out of elements delivered by a Source, and for constructing a TextDocument. A reader must accept following arguments in its signature:

- `elem` a named list with the components `content` and `uri` (as delivered by a Source via `getElem` or `pGetElem`).
- `language` a character string giving the language.
id a character giving a unique identifier for the created text document.

The element elem is typically provided by a source whereas the language and the identifier are normally provided by a corpus constructor (for the case that elem$content does not give information on these two essential items).

In case a reader expects configuration arguments we can use a function generator. A function generator is indicated by inheriting from class FunctionGenerator and function. It allows us to process additional arguments, store them in an environment, return a reader function with the well-defined signature described above, and still be able to access the additional arguments via lexical scoping. All corpus constructors in package tm check the reader function for being a function generator and if so apply it to yield the reader with the expected signature.

Value

For getReaders(), a character vector with readers provided by package tm.

See Also

readDOC, readPDF, readPlain, readRCV1, readRCV1asPlain, readReut21578XML, readReut21578XMLasPlain, and readXML.

readPDF

Description

Return a function which reads in a portable document format (PDF) document extracting both its text and its metadata.

Usage

readPDF(engine = c("pdftools", "xpdf", "Rpoppler", "ghostscript", "Rcampdf", "custom"), control = list(info = NULL, text = NULL))

Arguments

engine a character string for the preferred PDF extraction engine (see Details).
control a list of control options for the engine with the named components info and text (see Details).

Details

Formally this function is a function generator, i.e., it returns a function (which reads in a text document) with a well-defined signature, but can access passed over arguments (e.g., the preferred PDF extraction engine and control options) via lexical scoping.

Available PDF extraction engines are as follows.
"pdftools" (default) Poppler PDF rendering library as provided by the functions \texttt{pdf_info} and \texttt{pdf_text} in package \texttt{pdftools}.

"xpdf" command line pdfinfo and pdftotext executables which must be installed and accessible on your system. Suitable utilities are provided by the Xpdf (http://www.foolabs.com/xpdf/) PDF viewer or by the Poppler (http://poppler.freedesktop.org/) PDF rendering library.

"Rpoppler" Poppler PDF rendering library as provided by the functions \texttt{PDF_info} and \texttt{PDF_text} in package \texttt{Rpoppler}.

"ghostscript" Ghostscript using 'pdf_info.ps' and 'ps2ascii.ps'.

"Rcampdf" Perl CAM::PDF PDF manipulation library as provided by the functions \texttt{pdf_info} and \texttt{pdf_text} in package \texttt{Rcampdf}, available from the repository at http://datacube.wu.ac.at.

"custom" custom user-provided extraction engine.

Control parameters for engine "xpdf" are as follows.

\texttt{info} a character vector specifying options passed over to the \texttt{pdfinfo} executable.
\texttt{text} a character vector specifying options passed over to the \texttt{pdftotext} executable.

Control parameters for engine "custom" are as follows.

\texttt{info} a function extracting metadata from a PDF. The function must accept a file path as first argument and must return a named list with the components \texttt{Author} (as character string), \texttt{CreationDate} (of class \texttt{POSIXlt}), \texttt{Subject} (as character string), \texttt{Title} (as character string), and \texttt{Creator} (as character string).
\texttt{text} a function extracting content from a PDF. The function must accept a file path as first argument and must return a character vector.

\textbf{Value}

A function with the following formals:

\texttt{elem} a named list with the component \texttt{uri} which must hold a valid file name.
\texttt{language} a string giving the language.
\texttt{id} Not used.

The function returns a \texttt{PlainTextDocument} representing the text and metadata extracted from \texttt{elem$uri}.

\textbf{See Also}

\texttt{Reader} for basic information on the reader infrastructure employed by package \texttt{tm}.

\textbf{Examples}

\begin{verbatim}
uri <- sprintf("file://%s", system.file(file.path("doc", "tm.pdf"), package = "tm"))
pdf <- readPDF(elem = list(uri = uri), language = "en", id = "id1")
cat(content(pdf)[1])
VCorpus(URISource(uri, mode = ""),
   readerControl = list(reader = readPDF(engine = "ghostscript"))
\end{verbatim}
**readPlain**

**Read In a Text Document**

**Description**

Read in a text document without knowledge about its internal structure and possible available metadata.

**Usage**

```
readPlain(elem, language, id)
```

**Arguments**

- `elem` a named list with the component `content` which must hold the document to be read in.
- `language` a string giving the language.
- `id` a character giving a unique identifier for the created text document.

**Value**

A `PlainTextDocument` representing `elem$content`. The argument `id` is used as fallback if `elem$uri` is null.

**See Also**

`Reader` for basic information on the reader infrastructure employed by package `tm`.

**Examples**

```
docs <- c("This is a text.", "This another one.")
vs <- VectorSource(docs)
elem <- getElem(stepNext(vs))
(result <- readPlain(elem, "en", "id1"))
meta(result)
```

---

**readRCV1**

**Read In a Reuters Corpus Volume 1 Document**

**Description**

Read in a Reuters Corpus Volume 1 XML document.

**Usage**

```
readRCV1(elem, language, id)
readRCV1asPlain(elem, language, id)
```
Arguments

elem a named list with the component content which must hold the document to be read in.

language a string giving the language.

id Not used.

Value

An XMLTextDocument for readRCV1, or a PlainTextDocument for readRCV1asPlain, representing the text and metadata extracted from elem$content.

References


See Also

Reader for basic information on the reader infrastructure employed by package tm.

Examples

f <- system.file("texts", "rcv1_2330.xml", package = "tm")
f_bin <- readBin(f, raw(), file.info(f)$size)
rcv1 <- readRCV1(elem = list(content = f_bin), language = "en", id = "id1")
content(rcv1)
meta(rcv1)

Description

Read in a Reuters-21578 XML Document.

Usage

readReut21578XML(elem, language, id)
readReut21578XMLasPlain(elem, language, id)

Arguments

elem a named list with the component content which must hold the document to be read in.

language a string giving the language.

id Not used.
**readTagged**

**Value**

An `XMLTextDocument` for `readReut21578XML`, or a `PlainTextDocument` for `readReut21578XMLasPlain`, representing the text and metadata extracted from `elem$content`.

**References**


Luz, Saturnino *XML-encoded version of Reuters-21578*. [https://www.scss.tcd.ie/~luzs/tcs4114/sw/reuters21578-xml/](https://www.scss.tcd.ie/~luzs/tcs4114/sw/reuters21578-xml/)

**See Also**

*Reader* for basic information on the reader infrastructure employed by package *tm*.

---

**Description**

Return a function which reads in a text document containing POS-tagged words.

**Usage**

```r
readTagged(...) 
```

**Arguments**

`...` Arguments passed to `TaggedTextDocument`.

**Details**

Formally this function is a function generator, i.e., it returns a function (which reads in a text document) with a well-defined signature, but can access passed over arguments (`...`) via lexical scoping.

**Value**

A function with the following formals:

- `elem` a named list with the component `content` which must hold the document to be read in or the component `uri` holding a connection object or a character string.
- `language` a string giving the language.
- `id` a character giving a unique identifier for the created text document.

The function returns a `TaggedTextDocument` representing the text and metadata extracted from `elem$content` or `elem$uri`. The argument `id` is used as fallback if `elem$uri` is null.
See Also

Reader for basic information on the reader infrastructure employed by package tm.

Examples

```r
# See http://www.nltk.org/book/ch05.html or file ca01 in the Brown corpus
x <- paste("The/at grand/jj jury/nn commented/vbd on/in a/at number/nn of/in",
"other/ap topics/nns ./, among/in them/ppp the/at Atlanta/np and/cc",
"Fulton/np-tl County/nn-rl purchasing/vbg departments/nns which/wdt",
"it/pps said/vbd ```/``` are/ber well/qi operated/vbn and/cc follow/vb",
"generally/rb accepted/vbn practices/nns which/wdt inure/vb to/in the/at",
"best/jjt interest/nn of/in both/abx governments/nns ```/```/``` .```
)
vs <- VectorSource(x)
elem <- getElem(stepNext(vs))
(doc <- readTagged()(elem, language = "en", id = "id1"))
tagged_words(doc)
```

readXML

Read In an XML Document

Description

Return a function which reads in an XML document. The structure of the XML document is described with a specification.

Usage

```r
readXML(spec, doc)
```

Arguments

spec

A named list of lists each containing two components. The constructed reader will map each list entry to the content or metadatum of the text document as specified by the named list entry. Valid names include content to access the document’s content, and character strings which are mapped to metadata entries. Each list entry must consist of two components: the first must be a string describing the type of the second argument, and the second is the specification entry. Valid combinations are:

- `type = "node", spec = "XPathExpression"` The XPath (1.0) expression `spec` extracts information from an XML node.
- `type = "function", spec = function(doc) ...` The function `spec` is called, passing over the XML document (as delivered by `read_xml` from package xml2) as first argument.
- `type = "unevaluated", spec = "String"` The character vector `spec` is returned without modification.

doc

An (empty) document of some subclass of TextDocument.
Details

Formally this function is a function generator, i.e., it returns a function (which reads in a text document) with a well-defined signature, but can access passed over arguments (e.g., the specification) via lexical scoping.

Value

A function with the following formals:

- `elem` a named list with the component `content` which must hold the document to be read in.
- `language` a string giving the language.
- `id` a character giving a unique identifier for the created text document.

The function returns `doc` augmented by the parsed information as described by `spec` out of the XML file in `elem$content`. The arguments `language` and `id` are used as fallback: `language` if no corresponding metadata entry is found in `elem$content`, and `id` if no corresponding metadata entry is found in `elem$content` and if `elem$uri` is null.

See Also

- Reader for basic information on the reader infrastructure employed by package `tm`.
- Vignette 'Extensions: How to Handle Custom File Formats', and `XMLSource`.

---

### removeNumbers

| removeNumbers | Remove Numbers from a Text Document |

#### Description

Remove numbers from a text document.

#### Usage

```r
## S3 method for class 'character'
removeNumbers(x, ucp = FALSE, ...)
## S3 method for class 'PlainTextDocument'
removeNumbers(x, ...)
```

#### Arguments

- `x` a character vector or text document.
- `ucp` a logical specifying whether to use Unicode character properties for determining digit characters. If `FALSE` (default), characters in the ASCII `[:digit:]` class (i.e., the decimal digits from 0 to 9) are taken; if `TRUE`, the characters with Unicode general category `Nd` (Decimal_Number).
- `...` arguments to be passed to or from methods; in particular, from the `PlainTextDocument` method to the `character` method.
Value

The text document without numbers.

See Also

getTransformations to list available transformation (mapping) functions.
http://unicode.org/reports/tr44/#General_Category_Vvalues.

Examples

data("crude")
crude[[1]]
rmvouesNumbers(crude[[1]])

removePunctuation

Remove Punctuation Marks from a Text Document

Description

Remove punctuation marks from a text document.

Usage

```r
## S3 method for class 'character'
removePunctuation(x,
    preserve_intra_word_contractions = FALSE,
    preserve_intra_word_dashes = FALSE,
    ucp = FALSE, ...)

## S3 method for class 'PlainTextDocument'
removePunctuation(x, ...)
```

Arguments

- `x`: a character vector or text document.
- `preserve_intra_word_contractions`: a logical specifying whether intra-word contractions should be kept.
- `preserve_intra_word_dashes`: a logical specifying whether intra-word dashes should be kept.
- `ucp`: a logical specifying whether to use Unicode character properties for determining punctuation characters. If FALSE (default), characters in the ASCII [:punct:] class are taken; if TRUE, the characters with Unicode general category P (Punctuation).
- `...`: arguments to be passed to or from methods; in particular, from the PlainTextDocument method to the character method.
removeSparseTerms

Value
The character or text document `x` without punctuation marks (besides intra-word contractions (‘’)
and intra-word dashes (‘-‘) if `preserve_intra_word_contractions` and `preserve_intra_word_dashes`
are set, respectively).

See Also
`getTransformations` to list available transformation (mapping) functions.

Examples
```r
data("crude")
inspect(crude[[14]])
inspect(removePunctuation(crude[[14]]))
inspect(removePunctuation(crude[[14]],
                           preserve_intra_word_contractions = TRUE,
                           preserve_intra_word_dashes = TRUE))
```

---

### Description
Remove sparse terms from a document-term or term-document matrix.

### Usage
```
removeSparseTerms(x, sparse)
```

### Arguments

- `x` A `DocumentTermMatrix` or a `TermDocumentMatrix`.
- `sparse` A numeric for the maximal allowed sparsity in the range from bigger zero to smaller one.

### Value
A term-document matrix where those terms from `x` are removed which have at least a `sparse`
percentage of empty (i.e., terms occurring 0 times in a document) elements. I.e., the resulting
matrix contains only terms with a sparse factor of less than `sparse`.

### Examples
```r
data("crude")
tdm <- TermDocumentMatrix(crude)
removeSparseTerms(tdm, 0.2)
```
**removeWords**  
*Remove Words from a Text Document*

**Description**  
Remove words from a text document.

**Usage**  
```r  
## S3 method for class 'character'  
removeWords(x, words)  
## S3 method for class 'PlainTextDocument'  
removeWords(x, ...)  
```

**Arguments**  
- `x` A character or text document.
- `words` A character vector giving the words to be removed.
- `...` passed over argument `words`.

**Value**  
The character or text document without the specified words.

**See Also**  
- `getTransformations` to list available transformation (mapping) functions.
- `remove_stopwords` provided by package `tau`.

**Examples**  
```r  
data("crude")  
crude[[1]]  
removeWords(crude[[1]], stopwords("english"))  
```

---

**SimpleCorpus**  
*Simple Corpora*

**Description**  
Create simple corpora.

**Usage**  
```r  
SimpleCorpus(x, control = list(language = "en"))  
```
Arguments

x a `DataframeSource`, `DirSource` or `VectorSource`.

control a named list of control parameters.

language a character giving the language (preferably as IETF language tags, see `language` in package `NLP`). The default language is assumed to be English ("en").

Details

A simple corpus is fully kept in memory. Compared to a VCorpus, it is optimized for the most common usage scenario: importing plain texts from files in a directory or directly from a vector in R, preprocessing and transforming the texts, and finally exporting them to a term-document matrix. It adheres to the Corpus API. However, it takes internally various shortcuts to boost performance and minimize memory pressure; consequently it operates only under the following constraints:

- only `DataframeSource`, `DirSource` and `VectorSource` are supported,
- no custom readers, i.e., each document is read in and stored as plain text (as a string, i.e., a character vector of length one),
- transformations applied via `tm_map` must be able to process character vectors and return character vectors (of the same length),
- no lazy transformations in `tm_map`,
- no meta data for individual documents (i.e., no "local" in `meta`).

Value

An object inheriting from SimpleCorpus and Corpus.

See Also

Corpus for basic information on the corpus infrastructure employed by package tm.

VCorpus provides an implementation with volatile storage semantics, and PCorpus provides an implementation with permanent storage semantics.

Examples

txt <- system.file("texts", "txt", package = "tm")
(ovid <- SimpleCorpus(DirSource(txt, encoding = "UTF-8"),
                     control = list(language = "lat")))
Description

Creating and accessing sources.

Usage

```r
SimpleSource(encoding = "", length = 0,
position = 0,
reader = readPlain,
...,
class)
getSources()
## S3 method for class 'SimpleSource'
close(con, ...)
## S3 method for class 'SimpleSource'
eoi(x)
## S3 method for class 'DataframeSource'
getMeta(x)
## S3 method for class 'DataframeSource'
getElem(x)
## S3 method for class 'DirSource'
getElem(x)
## S3 method for class 'URISource'
getElem(x)
## S3 method for class 'VectorSource'
getElem(x)
## S3 method for class 'XMLSource'
getElem(x)
## S3 method for class 'SimpleSource'
length(x)
## S3 method for class 'SimpleSource'
open(con, ...)
## S3 method for class 'DataframeSource'
pGetElem(x)
## S3 method for class 'DirSource'
pGetElem(x)
## S3 method for class 'URISource'
pGetElem(x)
## S3 method for class 'VectorSource'
pGetElem(x)
## S3 method for class 'SimpleSource'
reader(x)
## S3 method for class 'SimpleSource'
```
stepNext(x)

Arguments

x A Source.

con A Source.

encoding a character giving the encoding of the elements delivered by the source.

length a non-negative integer denoting the number of elements delivered by the source. If the length is unknown in advance set it to 0.

position a numeric indicating the current position in the source.

reader a reader function (generator).

... For SimpleSource tag-value pairs for storing additional information; not used otherwise.

class a character vector giving additional classes to be used for the created source.

Details

Sources abstract input locations, like a directory, a connection, or simply an R vector, in order to acquire content in a uniform way. In packages which employ the infrastructure provided by package tm, such sources are represented via the virtual S3 class Source: such packages then provide S3 source classes extending the virtual base class (such as DirSource provided by package tm itself).

All extension classes must provide implementations for the functions close, eoi, getElem, length, open, reader, and stepNext. For parallel element access the (optional) function pGetElem must be provided as well. If document level metadata is available, the (optional) function getMeta must be implemented.

The functions open and close open and close the source, respectively. eoi indicates end of input. getElem fetches the element at the current position, whereas pGetElem retrieves all elements in parallel at once. The function length gives the number of elements. reader returns a default reader for processing elements. stepNext increases the position in the source to acquire the next element.

The function SimpleSource provides a simple reference implementation and can be used when creating custom sources.

Value

For SimpleSource, an object inheriting from class, SimpleSource, and Source.

For getSources, a character vector with sources provided by package tm.

open and close return the opened and closed source, respectively.

For eoi, a logical indicating if the end of input of the source is reached.

For getElem a named list with the components content holding the document and uri giving a uniform resource identifier (e.g., a file path or URL; NULL if not applicable or unavailable). For pGetElem a list of such named lists.

For length, an integer for the number of elements.

For reader, a function for the default reader.
See Also

DataframeSource, DirSource, URISource, VectorSource, and XMLSource.

stemCompletion

---

Complete Stems

Description

Heuristically complete stemmed words.

Usage

stemCompletion(x, dictionary, type = c("prevalent", "first", "longest", "none", "random", "shortest"))

Arguments

x A character vector of stems to be completed.
dictionary A Corpus or character vector to be searched for possible completions.
type A character naming the heuristics to be used:
prevalent Default. Takes the most frequent match as completion.
first Takes the first found completion.
longest Takes the longest completion in terms of characters.
none Is the identity.
random Takes some completion.
shortest Takes the shortest completion in terms of characters.

Value

A character vector with completed words.

References


Examples

data("crude")
stemCompletion(c("compan", "entit", "suppl"), crude)
stemDocument

Description
Stem words in a text document using Porter's stemming algorithm.

Usage

```
# S3 method for class 'character'
stemDocument(x, language = "english")
# S3 method for class 'PlainTextDocument'
stemDocument(x, language = meta(x, "language"))
```

Arguments

- `x`: A character vector or text document.
- `language`: A string giving the language for stemming.

Details
The argument `language` is passed over to `wordStem` as the name of the Snowball stemmer.

Examples

```
data("crude")
inspect(crude[[1]])
inspect(stemDocument(crude[[1]]))
```

stopwords

Description
Return various kinds of stopwords with support for different languages.

Usage

```
stopwords(kind = "en")
```

Arguments

- `kind`: A character string identifying the desired stopword list.
Details

Available stopword lists are:

catalan  Catalan stopwords (obtained from http://latel.upf.edu/morgana/altres/pub/ca_stop.htm),

romanian  Romanian stopwords (extracted from http://snowball.tartarus.org/otherapps/romanian/romanian1.tgz),

SMART  English stopwords from the SMART information retrieval system (as documented in Appendix 11 of http://jmlr.csail.mit.edu/papers/volume5/lewis04a/) (which coincides with the stopword list used by the MC toolkit (http://www.cs.utexas.edu/users/dml/software/mc/)),

and a set of stopword lists from the Snowball stemmer project in different languages (obtained from ‘http://svn.tartarus.org/snowball/trunk/website/algorithms/*/stop.txt’). Supported languages are danish, dutch, english, finnish, french, german, hungarian, italian, norwegian, portuguese, russian, spanish, and swedish. Language names are case sensitive. Alternatively, their IETF language tags may be used.

Value

A character vector containing the requested stopwords. An error is raised if no stopwords are available for the requested kind.

Examples

stopwords("en")
stopwords("SMART")
stopwords("german")

stripWhitespace  Strip Whitespace from a Text Document

Description

Strip extra whitespace from a text document. Multiple whitespace characters are collapsed to a single blank.

Usage

## S3 method for class 'PlainTextDocument'
stripWhitespace(x, ...)

Arguments

x  A text document.

...  Not used.
TermDocumentMatrix

Value

The text document with multiple whitespace characters collapsed to a single blank.

See Also

getTransformations to list available transformation (mapping) functions.

Examples

data("crude")
crude[[1]]
stripWhitespace(crude[[1]])

TermDocumentMatrix Term-Document Matrix

Description

Constructs or coerces to a term-document matrix or a document-term matrix.

Usage

TermDocumentMatrix(x, control = list())
DocumentTermMatrix(x, control = list())
as.TermDocumentMatrix(x, ...)
as.DocumentTermMatrix(x, ...)

Arguments

x a corpus for the constructors and either a term-document matrix or a document-term matrix or a simple triplet matrix (package slam) or a term frequency vector for the coercing functions.

control a named list of control options. There are local options which are evaluated for each document and global options which are evaluated once for the constructed matrix. Available local options are documented in termFreq and are internally delegated to a termFreq call.

This is different for a SimpleCorpus. In this case all options are processed in a fixed order in one pass to improve performance. It always uses the Boost (http://www.boost.org) Tokenizer (via Rcpp) and takes no custom functions as option arguments.

Available global options are:

bounds A list with a tag global whose value must be an integer vector of length 2. Terms that appear in less documents than the lower bound bounds$global[1] or in more documents than the upper bound bounds$global[2] are discarded. Defaults to list(global = c(1, Inf)) (i.e., every term will be used).
weighting A weighting function capable of handling a TermDocumentMatrix. It defaults to weightTf for term frequency weighting. Available weighting functions shipped with the tm package are weightTf, weightTfIdf, weightBin, and weightSMART.

... the additional argument weighting (typically a WeightFunction) is allowed when coercing a simple triplet matrix to a term-document or document-term matrix.

Value

An object of class TermDocumentMatrix or class DocumentTermMatrix (both inheriting from a simple triplet matrix in package slam) containing a sparse term-document matrix or document-term matrix. The attribute weighting contains the weighting applied to the matrix.

See Also
termFreq for available local control options.

Examples

data("crude")
tdm <- TermDocumentMatrix(crude, 
  control = list(removePunctuation = TRUE, 
                   stopwords = TRUE))
dtm <- DocumentTermMatrix(crude, 
  control = list(weighting = 
                 function(x) 
                 weightTfIdf(x, normalize = 
                               FALSE), 
                 stopwords = TRUE))
inspect(tdm[202:205, 1:5])
inspect(tdm[c("price", "prices", "texas"), c("127", "144", "191", "194")])
inspect(dtm[1:5, 273:276])

s <- SimpleCorpus(VectorSource(unlist(lapply(crude, as.character))))
m <- TermDocumentMatrix(s, 
  control = list(removeNumbers = TRUE, 
                 stopwords = TRUE, 
                 stemming = TRUE))
inspect(m[c("price", "texas"), c("127", "144", "191", "194")])
**termFreq**

### Usage

```r
termFreq(doc, control = list())
```

### Arguments

- **doc**
  An object inheriting from `TextDocument` or a character vector.

- **control**
  A list of control options which override default settings.

First, following two options are processed.

- **tokenize**
  A function tokenizing a `TextDocument` into single tokens, a `Span_Tokenizer`, `Token_Tokenizer`, or a string matching one of the predefined tokenization functions:
  - "Boost" for `boost_tokenizer`, or
  - "MC" for `MC_tokenizer`, or
  - "scan" for `scan_tokenizer`, or
  - "words" for `words`.
  Defaults to `words`.

- **tolower**
  Either a logical value indicating whether characters should be translated to lower case or a custom function converting characters to lower case. Defaults to `tolower`.

Next, a set of options which are sensitive to the order of occurrence in the control list. Options are processed in the same order as specified. User-specified options have precedence over the default ordering so that first all user-specified options and then all remaining options (with the default settings and in the order as listed below) are processed.

- **language**
  A character giving the language (preferably as IETF language tags, see `language` in package `NLP`) to be used for stopwords and stemming if not provided by `doc`.

- **removePunctuation**
  A logical value indicating whether punctuation characters should be removed from `doc`, a custom function which performs punctuation removal, or a list of arguments for `removePunctuation`. Defaults to `FALSE`.

- **removeNumbers**
  A logical value indicating whether numbers should be removed from `doc` or a custom function for number removal. Defaults to `FALSE`.

- **stopwords**
  Either a Boolean value indicating stopword removal using default language specific stopword lists shipped with this package, a character vector holding custom stopwords, or a custom function for stopword removal. Defaults to `FALSE`.

- **stemming**
  Either a Boolean value indicating whether tokens should be stemmed or a custom stemming function. Defaults to `FALSE`.

Finally, following options are processed in the given order.

- **dictionary**
  A character vector to be tabulated against. No other terms will be listed in the result. Defaults to `NULL` which means that all terms in `doc` are listed.
bounds A list with a tag local whose value must be an integer vector of length 2. Terms that appear less often in doc than the lower bound $\text{bounds}\{1\}$ or more often than the upper bound $\text{bounds}\{2\}$ are discarded. Defaults to $\text{list}(\text{local} = c(1, \text{Inf}))$ (i.e., every token will be used).

wordLengths An integer vector of length 2. Words shorter than the minimum word length $\text{wordLengths}\{1\}$ or longer than the maximum word length $\text{wordLengths}\{2\}$ are discarded. Defaults to $c(3, \text{Inf})$, i.e., a minimum word length of 3 characters.

Value
A table of class c("term_frequency", "table") with term frequencies as values and tokens as names.

See Also
getTokenizers

Examples
```r
data("crude")
termFreq(crude[[14]])
strsplit_space_tokenizer <- function(x)
  unlist(strsplit(as.character(x), "\[[:space:]\]+"))
ctrl <- list(tokenize = strsplit_space_tokenizer,
  removePunctuation = list(preserve_intra_word_dashes = TRUE),
  stopwords = c("reuter", "that"),
  stemming = TRUE,
  wordLengths = c(4, Inf))
termFreq(crude[[14]], control = ctrl)
```

Description
Representing and computing on text documents.

Details
Text documents are documents containing (natural language) text. The \texttt{tm} package employs the infrastructure provided by package \texttt{NLP} and represents text documents via the virtual S3 class \texttt{TextDocument}. Actual S3 text document classes then extend the virtual base class (such as \texttt{PlainTextDocument}). All extension classes must provide an \texttt{as.character} method which extracts the natural language text in documents of the respective classes in a “suitable” (not necessarily structured) form, as well as \texttt{content} and \texttt{meta} methods for accessing the (possibly raw) document content and metadata.
**Description**

Combine several corpora into a single one, combine multiple documents into a corpus, combine multiple term-document matrices into a single one, or combine multiple term frequency vectors into a single term-document matrix.

**Usage**

```r
## S3 method for class 'VCorpus'
c(..., recursive = FALSE)
## S3 method for class 'TextDocument'
c(..., recursive = FALSE)
## S3 method for class 'TermDocumentMatrix'
c(..., recursive = FALSE)
## S3 method for class 'term_frequency'
c(..., recursive = FALSE)
```

**Arguments**

- `...` Corpora, text documents, term-document matrices, or term frequency vectors.
- `recursive` Not used.

**See Also**

`VCorpus`, `TextDocument`, `TermDocumentMatrix`, and `termFreq`.

**Examples**

```r
data("acq")
data("crude")
meta(acq, "comment", type = "corpus") <- "Acquisitions"
meta(crude, "comment", type = "corpus") <- "Crude oil"
meta(acq, "acqLabels") <- 1:50
meta(acq, "jointLabels") <- 1:50
meta(crude, "crudeLabels") <- letters[1:20]
meta(crude, "jointLabels") <- 1:20
c(acq, crude)
meta(c(acq, crude), type = "corpus")
```
tm_filter

Filter and Index Functions on Corpora

Description

Interface to apply filter and index functions to corpora.

Usage

```r
## S3 method for class 'PCorpus'
tm_filter(x, FUN, ...)
## S3 method for class 'SimpleCorpus'
tm_filter(x, FUN, ...)
## S3 method for class 'VCorpus'
tm_filter(x, FUN, ...)
## S3 method for class 'PCorpus'
tm_index(x, FUN, ...)
## S3 method for class 'SimpleCorpus'
tm_index(x, FUN, ...)
## S3 method for class 'VCorpus'
tm_index(x, FUN, ...)
```

Arguments

- **x**: A corpus.
- **FUN**: a filter function taking a text document or a string (if `x` is a `SimpleCorpus`) as input and returning a logical value.
- **...**: arguments to `FUN`.

Value

- `tm_filter` returns a corpus containing documents where `FUN` matches, whereas `tm_index` only returns the corresponding indices.

Examples

```r
data("crude")
# Full-text search
tm_filter(crude, FUN = function(x) any(grep("co[m]?pany", content(x))))
```
Description

Interface to apply transformation functions (also denoted as mappings) to corpora.

Usage

```r
## S3 method for class 'PCorpus'
tm_map(x, FUN, ...)
## S3 method for class 'SimpleCorpus'
tm_map(x, FUN, ...)
## S3 method for class 'VCorpus'
 tm_map(x, FUN, ..., lazy = FALSE)
```

Arguments

- `x` A corpus.
- `FUN` a transformation function taking a text document (a character vector when `x` is a `SimpleCorpus`) as input and returning a text document (a character vector of the same length as the input vector for `SimpleCorpus`). The function `content_transformer` can be used to create a wrapper to get and set the content of text documents.
- `...` arguments to `FUN`.
- `lazy` a logical. Lazy mappings are mappings which are delayed until the content is accessed. It is useful for large corpora if only few documents will be accessed. In such a case it avoids the computationally expensive application of the mapping to all elements in the corpus.

Value

A corpus with `FUN` applied to each document in `x`. In case of lazy mappings only internal flags are set. Access of individual documents triggers the execution of the corresponding transformation function.

Note

Lazy transformations change R’s standard evaluation semantics.

See Also

- `getTransformations` for available transformations.
Examples

data("crude")
## Document access triggers the stemming function
## (i.e., all other documents are not stemmed yet)
tm_map(crude, stemDocument, lazy = TRUE)[[1]]
## Use wrapper to apply character processing function
tm_map(crude, content_transformer(tolower))
## Generate a custom transformation function which takes the heading as new content
headings <- function(x)
  PlainTextDocument(meta(x, "heading"),
    id = meta(x, "id"),
    language = meta(x, "language"))
inspect(tm_map(crude, headings))

tm_reduce

Combine Transformations

Description

Fold multiple transformations (mappings) into a single one.

Usage

tm_reduce(x, tmFuns, ...)

Arguments

  x  A corpus.
  tmFuns  A list of tm transformations.
  ...  Arguments to the individual transformations.

Value

  A single tm transformation function obtained by folding tmFuns from right to left (via Reduce(..., right = TRUE)).

See Also

  Reduce for R’s internal folding/accumulation mechanism, and getTransformations to list available transformation (mapping) functions.

Examples

data(crude)
crude[[1]]
skipWords <- function(x) removeWords(x, c("it", "the"))
funs <- list(stripWhitespace, skipWords, removePunctuation, content_transformer(tolower))
tm_map(crude, FUN = tm_reduce, tmFuns = funs)[[1]]
tm_term_score  

Compute Score for Matching Terms

Description

Compute a score based on the number of matching terms.

Usage

```r
## S3 method for class 'DocumentTermMatrix'
tm_term_score(x, terms, FUN = row_sums)
## S3 method for class 'PlainTextDocument'
tm_term_score(x, terms, FUN = function(x) sum(x, na.rm = TRUE))
## S3 method for class 'term_frequency'
(tm_term_score(x, terms, FUN = function(x) sum(x, na.rm = TRUE)))
## S3 method for class 'TermDocumentMatrix'
(tm_term_score(x, terms, FUN = col_sums))
```

Arguments

- `x`: Either a `PlainTextDocument`, a term frequency as returned by `termFreq`, or a `TermDocumentMatrix`.
- `terms`: A character vector of terms to be matched.
- `FUN`: A function computing a score from the number of terms matching in `x`.

Value

A score as computed by `FUN` from the number of matching terms in `x`.

Examples

```r
data("acq")
tm_term_score(acq[[1]], c("company", "change"))
## Not run: ## Test for positive and negative sentiments
## install.packages("tm.lexicon.GeneralInquirer", repos="http://datacube.wu.ac.at", type="source")
require("tm.lexicon.GeneralInquirer")
sapply(acq[1:10], tm_term_score, terms_in_General_Inquirer_categories("Positiv"))
sapply(acq[1:10], tm_term_score, terms_in_General_Inquirer_categories("Negativ"))
tm_term_score(TermDocumentMatrix(acq[1:10],
                   control = list(removePunctuation = TRUE)),
            terms_in_General_Inquirer_categories("Positiv"))
## End(Not run)
```
tokenizer

Tokenizers

Description

Tokenize a document or character vector.

Usage

Boost_tokenizer(x)
MC_tokenizer(x)
scan_tokenizer(x)

Arguments

x  A character vector, or an object that can be coerced to character by as.character.

Details

The quality and correctness of a tokenization algorithm highly depends on the context and application scenario. Relevant factors are the language of the underlying text and the notions of whitespace (which can vary with the used encoding and the language) and punctuation marks. Consequently, for superior results you probably need a custom tokenization function.

Boost_tokenizer  Uses the Boost (http://www.boost.org) Tokenizer (via Rcpp).
MC_tokenizer  Implements the functionality of the tokenizer in the MC toolkit (http://www.cs.utexas.edu/users/dml/software/mc/).
scan_tokenizer  Simulates scan(..., what = "character").

Value

A character vector consisting of tokens obtained by tokenization of x.

See Also

getTokenizers to list tokenizers provided by package tm.
Regexp_Tokenizer for tokenizers using regular expressions provided by package NLP.
tokenize for a simple regular expression based tokenizer provided by package tau.
tokenizers for a collection of tokenizers provided by package tokenizers.
Examples

```r
data("crude")
Boost_tokenizer(crude[[1]])
MC_tokenizer(crude[[1]])
scan_tokenizer(crude[[1]])
strsplit_space_tokenizer <- function(x)
  unlist(strsplit(as.character(x), "\[[:space:]]+\))
strsplit_space_tokenizer(crude[[1]])
```
vcorpus

Examples

loremipsum <- system.file("texts", "loremipsum.txt", package = "tm")
ovid <- system.file("texts", "txt", "ovid_1.txt", package = "tm")
us <- URIsource(sprintf("file://%s", c(loremipsum, ovid)))
inspect(vcorpus(us))

---

VCorpus

Volatile Corpora

Description

Create volatile corpora.

Usage

VCorpus(x, readerControl = list(reader = reader(x), language = "en"))
as.VCorpus(x)

Arguments

x
readerControl
reader
language

For VCorpus a Source object, and for as.VCorpus an R object.
a named list of control parameters for reading in content from x.
a function capable of reading in and processing the format delivered by x.
a character giving the language (preferably as IETF language tags, see
language in package NLP). The default language is assumed to be English ("en").

Details

A volatile corpus is fully kept in memory and thus all changes only affect the corresponding R object.

Value

An object inheriting from VCorpus and Corpus.

See Also

Corpus for basic information on the corpus infrastructure employed by package tm.
PCorpus provides an implementation with permanent storage semantics.

Examples

reut21578 <- system.file("texts", "crude", package = "tm")
VCorpus(DirSource(reut21578, mode = "binary"),
list(reader = readReut21578XMLasPlain))
**VectorSource**

Description

Create a vector source.

Usage

VectorSource(x)

Arguments

x  
A vector giving the texts.

Details

A *vector source* interprets each element of the vector *x* as a document.

Value

An object inheriting from VectorSource, SimpleSource, and Source.

See Also

Source for basic information on the source infrastructure employed by package tm.

Examples

docs <- c("This is a text.", "This another one.")
(vs <- VectorSource(docs))
inspect(VCorpus(vs))

**weightBin**

Description

Binary weight a term-document matrix.

Usage

weightBin(m)

Arguments

m  
A TermDocumentMatrix in term frequency format.
Details

Formally this function is of class WeightingFunction with the additional attributes name and acronym.

Value

The weighted matrix.

Description

Construct a weighting function for term-document matrices.

Usage

WeightFunction(x, name, acronym)

Arguments

x A function which takes a TermDocumentMatrix with term frequencies as input, weights the elements, and returns the weighted matrix.
name A character naming the weighting function.
acronym A character giving an acronym for the name of the weighting function.

Value

An object of class WeightFunction which extends the class function representing a weighting function.

Examples

weightCutBin <- WeightFunction(function(m, cutoff) m > cutoff,
"binary with cutoff", "bincut")
weightSMART

SMART Weightings

Description

Weight a term-document matrix according to a combination of weights specified in SMART notation.

Usage

weightSMART(m, spec = "nnn", control = list())

Arguments

m A TermDocumentMatrix in term frequency format.
spec a character string consisting of three characters. The first letter specifies a term frequency schema, the second a document frequency schema, and the third a normalization schema. See Details for available built-in schemata.
control a list of control parameters. See Details.

Details

Formally this function is of class WeightingFunction with the additional attributes name and acronym.

The first letter of spec specifies a weighting schema for term frequencies of m:

"n" (natural) \( t_{i,j} \) counts the number of occurrences \( n_{i,j} \) of a term \( t_i \) in a document \( d_j \). The input term-document matrix m is assumed to be in this standard term frequency format already.

"l" (logarithm) is defined as \( 1 + \log_2(t_{i,j}) \).

"a" (augmented) is defined as \( 0.5 + \frac{0.5 + t_{i,j}}{\max_i(t_{i,j})} \).

"b" (boolean) is defined as 1 if \( t_{i,j} > 0 \) and 0 otherwise.

"L" (log average) is defined as \( \frac{1 + \log_2(t_{i,j})}{1 + \log_2(\text{ave}_{e \in j}(t_{i,e}))} \).

The second letter of spec specifies a weighting schema of document frequencies for m:

"n" (no) is defined as 1.

"t" (idf) is defined as \( \log_2 \frac{N}{df_t} \) where \( df_t \) denotes how often term \( t \) occurs in all documents.

"p" (prob idf) is defined as \( \max(0, \log_2(\frac{N - df_t}{df_t})) \).

The third letter of spec specifies a schema for normalization of m:

"n" (none) is defined as 1.

"c" (cosine) is defined as \( \sqrt{\text{col_sums}(m^2)} \).
"u" (pivoted unique) is defined as $slope \times \sqrt{\text{col_sums}(m^2)} + (1 - slope) \times pivot$ where both $slope$ and $pivot$ must be set via named tags in the control list.

"b" (byte size) is defined as $\frac{1}{\text{CharLength}^\alpha}$. The parameter $\alpha$ must be set via the named tag $\text{alpha}$ in the control list.

The final result is defined by multiplication of the chosen term frequency component with the chosen document frequency component with the chosen normalization component.

Value
The weighted matrix.

References

Examples
data("crude")
TermDocumentMatrix(crude,
   control = list(removePunctuation = TRUE,
                  stopwords = TRUE,
                  weighting = function(x)
                      weightSMART(x, spec = "ntc")))

weightTf  Weight by Term Frequency

Description
Weight a term-document matrix by term frequency.

Usage
weightTf(m)

Arguments
m  A *TermDocumentMatrix* in term frequency format.

Details
Formally this function is of class *WeightingFunction* with the additional attributes *name* and *acronym*.

This function acts as the identity function since the input matrix is already in term frequency format.

Value
The weighted matrix.
weightTfIdf

Weight by Term Frequency - Inverse Document Frequency

Description

Weight a term-document matrix by term frequency - inverse document frequency.

Usage

weightTfIdf(m, normalize = TRUE)

Arguments

- **m**: A `TermDocumentMatrix` in term frequency format.
- **normalize**: A Boolean value indicating whether the term frequencies should be normalized.

Details

Formally this function is of class `WeightingFunction` with the additional attributes name and acronym.

Term frequency $tf_{i,j}$ counts the number of occurrences $n_{i,j}$ of a term $t_i$ in a document $d_j$. In the case of normalization, the term frequency $tf_{i,j}$ is divided by $\sum_k n_{k,j}$.

Inverse document frequency for a term $t_i$ is defined as

$$idf_i = \log_2 \frac{|D|}{|\{d \mid t_i \in d\}|}$$

where $|D|$ denotes the total number of documents and where $|\{d \mid t_i \in d\}|$ is the number of documents where the term $t_i$ appears.

Term frequency - inverse document frequency is now defined as $tf_{i,j} \cdot idf_i$.

Value

The weighted matrix.

References

writeCorpus  
*Write a Corpus to Disk*

**Description**

Write a plain text representation of a corpus to multiple files on disk corresponding to the individual documents in the corpus.

**Usage**

```
writeCorpus(x, path = ".", filenames = NULL)
```

**Arguments**

- `x`: A corpus.
- `path`: A character listing the directory to be written into.
- `filenames`: Either NULL or a character vector. In case no filenames are provided, filenames are automatically generated by using the documents’ identifiers in `x`.

**Details**

The plain text representation of the corpus is obtained by calling `as.character` on each document.

**Examples**

```
data("crude")
## Not run: writeCorpus(crude, path = ".", filenames = paste(seq_along(crude), ".txt", sep = ""))
## End(Not run)
```

---

**XMLSource**

*XML Source*

**Description**

Create an XML source.

**Usage**

```
XMLSource(x, parser = xml_contents, reader)
```
XMLTextDocument

Arguments

- **x**: A character giving a uniform resource identifier.
- **parser**: A function accepting an XML document (as delivered by `read_xml` in package `xml2`) as input and returning XML elements/nodes.
- **reader**: A function capable of turning XML elements/nodes as returned by `parser` into a subclass of `TextDocument`.

Value

An object inheriting from `XMLSource`, `SimpleSource`, and `Source`.

See Also

- `Source` for basic information on the source infrastructure employed by package `tm`.
- Vignette 'Extensions: How to Handle Custom File Formats', and `readXML`.

---

**XMLTextDocument**  
**XML Text Documents**

### Description

Create XML text documents.

### Usage

```r
XMLTextDocument(x = xml_missing(),
    author = character(0),
    datetimestamp = as.POSIXlt(Sys.time(), tz = "GMT"),
    description = character(0),
    heading = character(0),
    id = character(0),
    language = character(0),
    origin = character(0),
    ...,
    meta = NULL)
```

### Arguments

- **x**: An `XMLDocument`.
- **author**: A character or an object of class `person` giving the author names.
- **datetimestamp**: An object of class `POSIXt` or a character string giving the creation date/time information. If a character string, exactly one of the ISO 8601 formats defined by `http://www.w3.org/TR/NOTE-datetime` should be used. See `parse_ISO_8601_datetime` in package `NLP` for processing such date/time information.
- **description**: A character giving a description.
heading a character giving the title or a short heading.
id a character giving a unique identifier.
language a character giving the language (preferably as IETF language tags, see language in package NLP).
origin a character giving information on the source and origin.
... user-defined document metadata tag-value pairs.
meta a named list or NULL (default) giving all metadata. If set all other metadata arguments are ignored.

Value
An object inheriting from XMLTextDocument and TextDocument.

See Also
TextDocument for basic information on the text document infrastructure employed by package tm.

Examples
xml <- system.file("extdata", "order-doc.xml", package = "xml2")
(xtd <- XMLTextDocument(xml2::read_xml(xml),
  heading = "XML text document",
  id = xml,
  language = "en")
content(xtd)
meta(xtd)

Zipf_n_Heaps Explore Corpus Term Frequency Characteristics

Description
Explore Zipf’s law and Heaps’ law, two empirical laws in linguistics describing commonly observed characteristics of term frequency distributions in corpora.

Usage
Zipf_plot(x, type = "l", ...)
Heaps_plot(x, type = "l", ...)

Arguments
x a document-term matrix or term-document matrix with unweighted term frequencies.
type a character string indicating the type of plot to be drawn, see plot.
... further graphical parameters to be used for plotting.
Zipf’s law (e.g., http://en.wikipedia.org/wiki/Zipf%27s_law) states that given some corpus of natural language utterances, the frequency of any word is inversely proportional to its rank in the frequency table, or, more generally, that the pmf of the term frequencies is of the form $ck^{-\beta}$, where $k$ is the rank of the term (taken from the most to the least frequent one). We can conveniently explore the degree to which the law holds by plotting the logarithm of the frequency against the logarithm of the rank, and inspecting the goodness of fit of a linear model.

Heaps’ law (e.g., http://en.wikipedia.org/wiki/Heaps%27_law) states that the vocabulary size $V$ (i.e., the number of different terms employed) grows polynomially with the text size $T$ (the total number of terms in the texts), so that $V = cT^\beta$. We can conveniently explore the degree to which the law holds by plotting $\log(V)$ against $\log(T)$, and inspecting the goodness of fit of a linear model.

Value

The coefficients of the fitted linear model. As a side effect, the corresponding plot is produced.

Examples

```r
data("acq")
m <- DocumentTermMatrix(acq)
Zipf_plot(m)
Heaps_plot(m)
```

Description

Create a ZIP file source.

Usage

```r
ZipSource(zipfile, 
  pattern = NULL, 
  recursive = FALSE, 
  ignore.case = FALSE, 
  mode = "text")
```  

Arguments

- `zipfile` A character string with the full path name of a ZIP file.
- `pattern` an optional regular expression. Only file names in the ZIP file which match the regular expression will be returned.
- `recursive` logical. Should the listing recurse into directories?
- `ignore.case` logical. Should pattern-matching be case-insensitive?
mode

a character string specifying if and how files should be read in. Available modes are:

"" No read. In this case `getElem` and `pGetElem` only deliver URIs.
"binary" Files are read in binary raw mode (via `readBin`).
"text" Files are read as text (via `readLines`).

Details

A ZIP file source extracts a compressed ZIP file via `unzip` and interprets each file as a document.

Value

An object inheriting from `ZipSource`, `SimpleSource`, and `Source`.

See Also

`Source` for basic information on the source infrastructure employed by package `tm`.

Examples

```r
zipfile <- tempfile()
files <- Sys.glob(file.path(system.file("texts", "txt", package = "tm"), "*"))
zip(zipfile, files)
zipfile <- paste0(zipfile, ".zip")
Corpus(ZipSource(zipfile, recursive = TRUE))[1]
file.remove(zipfile)
```
Index

*Topic IO
  foreign, 11
*Topic datasets
  acq, 3
  crude, 5
*Topic file
  readPDF, 23
  stopwords, 37
*Topic math
  termFreq, 40
  \[, 5\]
  \[[], 5\]
  acq, 3
  antiword, 22
  as.character, 42
  as.DocumentTermMatrix
    (TermDocumentMatrix), 39
  as.list, 5
  as.TermDocumentMatrix
    (TermDocumentMatrix), 39
  as.VCorpus (VCorpus), 50
Boost_tokenizer, 12, 41
Boost_tokenizer (tokenizer), 48
  c.term_frequency (tm_combine), 43
  c.TermDocumentMatrix (tm_combine), 43
  c.TextDocument (tm_combine), 43
  c.VCorpus (tm_combine), 43
  close.SimpleSource (Source), 34
  content, 4, 42
  content_transformer, 4, 13, 45
  Corpus, 4, 16, 18, 33, 36, 50
  crude, 5

  DataframeSource, 6, 33, 36
  DCorpus, 5
  dir, 7
  DirSource, 7, 33, 35, 36

  Docs, 8
  document-term matrix, 12
  DocumentTermMatrix, 8–10, 31
  DocumentTermMatrix
    (TermDocumentMatrix), 39
  DublinCore (meta), 15
  DublinCore<- (meta), 15
  Encoding, 8, 49
  eoi (Source), 34
  filehashOption, 17
  findAssocs, 9
  findFreqTerms, 9
  findMostFreqTerms, 10
  foreign, 11
  FunctionGenerator (Reader), 22
  getElem, 7, 22, 49, 60
  getElem (Source), 34
  getMeta (Source), 34
  getReaders (Reader), 22
  getSources (Source), 34
  getTokenizers, 12, 42, 48
  getTransformations, 13, 30–32, 39, 45, 46
  graphNEL, 20
  Heaps_plot (Zipf_n_Heaps), 58
  hpc, 13
  iconv, 7, 8, 49
  inspect, 15
  language, 17, 19, 33, 41, 50, 58
  lapply, 14
  length, 5
  length.SimpleSource (Source), 34
  makeCluster, 13, 14
  makeForkCluster, 14
  MC_tokenizer, 12, 41
INDEX

MC_tokenizer (tokenizer), 48
mclapply, 14
meta, 5, 6, 15, 16, 33, 42
meta<-.PCorpus (meta), 15
meta<-.PlainTextDocument (meta), 15
meta<-.SimpleCorpus (meta), 15
meta<-.XMLTextDocument (meta), 15

nDocs (Docs), 8
nTerms (Docs), 8

open.SimpleSource (Source), 34

parLapply, 14
parLapplyLB, 14
parse_ISO8601_datetime, 19, 57
PCorpus, 5, 14, 17, 33, 50
PDF_info, 24
pdf_info, 24
PDF_text, 24
pdf_text, 24
person, 19, 57
pGetElem, 7, 22, 49, 60
pGetElem (Source), 34
PlainTextDocument, 18, 21, 22, 24–27, 42, 43, 47
plot, 19, 58
POSIXt, 19, 57

read_dtm_Blei_et_al (foreign), 11
read_dtm_MC (foreign), 11
read_stm_MC, 12
read_xml, 28, 57
readBin, 7, 49, 60
readDataframe, 20
readDOC, 21, 23
Reader, 21, 22, 22, 24–29
reader (Source), 34
readLines, 7, 49, 60
readPDF, 23, 23
readPlain, 23, 25
readRCV1, 23, 25
readRCV1asPlain, 23
readRCV1asPlain (readRCV1), 25
readReut21578XXML, 23, 26
readReut21578XMLasPlain, 23
readReut21578XMLasPlain (readReut21578XML), 26
readTagged, 27
readtext, 6
readXML, 23, 28, 57
regex, 31
Regexp_Tokenizer, 48
remove_stopwords, 32
removeNumbers, 13, 29
removePunctuation, 13, 30, 41
removeSparseTerms, 31
removeWords, 13, 32

scan_tokenizer, 12, 41
scan_tokenizer (tokenizer), 48
setDefaultCluster, 14
simple triplet matrix, 39, 40
SimpleCorpus, 5, 32, 39
SimpleSource, 6, 8, 49, 51, 57, 60
SimpleSource (Source), 44
Source, 6, 8, 17, 22, 34, 49–51, 57, 60
Span_Tokenizer, 41
stemCompletion, 36
stemDocument, 13, 37
stepNext (Source), 34
stopwords, 37
stripWhitespace, 13, 38

TaggedTextDocument, 27
term frequency vector, 39
TermDocumentMatrix, 8–10, 14, 31, 39, 43, 47, 51–55
termFreq, 10, 39, 40, 40, 43, 47
Terms (Docs), 8
TextDocument, 16, 19, 22, 41, 42, 43, 57, 58
tm_combine, 43
tm_filter, 44
tm_index, 14
tm_index (tm_filter), 44
tm_map, 4, 13, 14, 33, 45
tm_parLapply (hpc), 13
tm_parLapply_engine (hpc), 13
tm_reduce, 46
tm_term_score, 47
Token_Tokenizer, 41
tokenize, 48
tokenizer, 48
tokenizers, 48
tolower, 41

unzip, 60
INDEX

URI Source, 36, 49

VCorpus, 3, 5, 14, 18, 33, 43, 50
Vector Source, 33, 36, 51

weight Bin, 40, 51
Weight Function, 40, 52
weight SMART, 40, 53
weight TF, 40, 54
weight TF IDF, 40, 55
words, 41
word Stem, 37
write Corpus, 56

XML Document, 57
XML Source, 29, 36, 56
XML Text Document, 26, 27, 43, 57

Zipf n Heaps, 58
Zipf plot (Zipf n Heaps), 58
Zip Source, 59