Package ‘tm’

February 20, 2015

Title Text Mining Package
Version 0.6
Date 2014-06-06
Depends R (>= 3.1.0), NLP (>= 0.1-2)
Imports parallel, slam (>= 0.1-31), stats, tools, utils
Suggests filehash, Rcampdf, Rgraphviz, Rpoppler, SnowballC,
tm.lexicon.GeneralInquirer, XML
SystemRequirements Antiword (http://www.winfield.demon.nl/) for
reading MS Word files, pdfinfo and pdftotext from Poppler
(http://poppler.freedesktop.org/) for reading PDF
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
Author Ingo Feinerer [aut, cre],
Kurt Hornik [aut],
Artifex Software, Inc. [ctb, cph] (pdf_info.ps taken from GPL
Ghostscript)
Maintainer Ingo Feinerer <feinerer@logic.at>
NeedsCompilation yes
Repository CRAN
Date/Publication 2014-06-11 15:06:08

R topics documented:

acq ................................................................. 3
content_transformer ........................................... 4
Corpus ............................................................. 4
crude ............................................................. 5
DataframeSource .............................................. 6
R topics documented:

- DirSource .................................................. 7
- Docs .......................................................... 8
- findAssocs .................................................. 9
- findFreqTerms .............................................. 9
- foreign ....................................................... 10
- getTokenizers ................................................ 11
- getTransformations ........................................ 12
- inspect ....................................................... 12
- meta .......................................................... 13
- FCorpus ...................................................... 14
- PlainTextDocument ......................................... 15
- plot ........................................................... 17
- readDOC ...................................................... 18
- Reader ......................................................... 19
- readPDF ....................................................... 20
- readPlain ..................................................... 21
- readRCV1 ..................................................... 22
- readReut21578XML .......................................... 23
- readTabular .................................................. 24
- readXML ....................................................... 25
- removeNumbers ............................................... 26
- removePunctuation .......................................... 27
- removeSparseTerms ......................................... 28
- removeWords ................................................ 28
- Source ......................................................... 29
- stemCompletion ............................................. 31
- stemDocument ............................................... 32
- stopwords ................................................... 33
- stripWhitespace ............................................. 34
- TermDocumentMatrix ........................................ 34
- termFreq ...................................................... 36
- TextDocument ................................................ 37
- tm_combine .................................................. 38
- tm_filter ..................................................... 39
- tm_map ......................................................... 40
- tm_reduce .................................................... 41
- tm_term_score ............................................... 42
- tokenizer ..................................................... 43
- URISource ................................................... 44
- VCorpus ....................................................... 45
- VectorSource ............................................... 46
- weightBin ..................................................... 46
- WeightFunction .............................................. 47
- weightSMART ................................................ 48
- weightTf ...................................................... 49
- weightTfIdf .................................................. 50
- writeCorpus .................................................. 51
- XMLSource ................................................... 51
50 Exemplary News Articles from the Reuters-21578 Data Set of Topic acq

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


Examples
data("acq")
acq
**content_transformer**  

**Content Transformers**

**Description**

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

**Usage**

```r
current_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**

```r
data("crude")
crude[[1]]
(f <- current_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).

**See Also**

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

*D Corpus* for a distributed corpus class provided by package *tm.plugin.dc*.

| crude | 20 Exemplary News Articles from the Reuters-21578 Data Set of Topic crude |

**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**


Luz, Saturnino *XML-encoded version of Reuters-21578*. [http://modnlp.berlios.de/reuters21578.html](http://modnlp.berlios.de/reuters21578.html)
DataframeSource

Examples

```r
data("crude")
crude
```

Description

Create a data frame source.

Usage

```r
DataframeSource(x)
```

Arguments

- `x`: A data frame giving the texts.

Details

A `data frame source` interprets each row of the data frame `x` as a document.

Value

An object inheriting from DataframeSource, SimpleSource, and Source.

See Also

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

Examples

```r
docs <- data.frame(c("This is a text.", "This another one."))
(ds <- DataframeSource(docs))
inspect(VCorpus(ds))
```
DirSource

Directory Source

Description

Create a directory source.

Usage

\[
\text{DirSource(directory = "}", \\
\quad \text{encoding = "}", \\
\quad \text{pattern = NULL,} \\
\quad \text{recursive = FALSE,} \\
\quad \text{ignore.case = FALSE,} \\
\quad \text{mode = } \text{"text"})
\]

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 convert the input to UTF-8.
- **pattern**: an optional regular expression. Only file names 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 `getElement` and `pGetElement` 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.
Access Document IDs and Terms

**Description**

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

**Usage**

```r
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**

```r
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

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

---

**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).
Tokenizers

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_dtm_MC in package slam.

Description

Predefined tokenizers.

Usage

getTokenizers()

Value

A character vector with tokenizers provided by package tm.

See Also

MC_tokenizer and scan_tokenizer.

Examples

getTokenizers()
getTransformations

Description
Predefined transformations (mappings) which can be used with \texttt{tm_map}.

Usage
getTransformations()

Value
A character vector with transformations provided by package \texttt{tm}.

See Also
\texttt{removeNumbers}, \texttt{removePunctuation}, \texttt{removeWords}, \texttt{stemDocument}, and \texttt{stripWhitespace}.\texttt{content_transformer} to create custom transformations.

Examples
getTransformations()

inspect

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

Usage
\begin{verbatim}
## S3 method for class 'PCorpus'
inspect(x)
## S3 method for class 'VCorpus'
inspect(x)
## S3 method for class 'TermDocumentMatrix'
inspect(x)
\end{verbatim}

Arguments
\begin{itemize}
\item [x] Either a corpus or a term-document matrix.
\end{itemize}
Examples

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

Description

Accessing and modifying metadata of text documents and corpora.

Usage

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

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).
- **...**: Not used.
- **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 form 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/).

References


See Also

meta for metadata in package NLP.

Examples

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**

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 \( x \).
- `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`.

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

```r
 txt <- system.file("texts", "txt", package = "tm")
## Not run: PCorpus(DirSource(txt),
    dbControl = list(dbName = "pcorpus.db", dbType = "DB1"))
## End(Not run)
```

Description

Create plain text documents.
Usage

```r
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.
- **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()), 
```
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)

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

plot(tdm, corThreshold = 0.2, weighting = TRUE)
## End(Not run)
```
readDOC  Read In a MS Word Document

Description

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

Usage

    readDOC(AntiwordOptions = "")

Arguments

    AntiwordOptions
        Options passed over to antiword.

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.

    Note that this MS Word reader needs the tool antiword 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/.

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.
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, readTabular, and readXML.
**readPDF**  
*Read In a PDF Document*

**Description**

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

**Usage**

```r
readPDF(engine = c("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.

- **"xpdf"** (default) 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/](http://www.foolabs.com/xpdf/)) PDF viewer or by the Poppler ([http://poppler.freedesktop.org/](http://poppler.freedesktop.org/)) PDF rendering library.

- **"Rpoppler"** Poppler PDF rendering library as provided by the functions PDF_info and PDF_text in package **Rpoppler**.

- **"ghostscript"** Ghostscript using ‘pdf_info.ps’ and ‘ps2ascii.ps’.

- **"Rcampdf"** Perl CAM::PDF PDF manipulation library as provided by the functions pdf_info and pdf_text in package **Rcampdf**, available from the repository at [http://datacube.wu.ac.at](http://datacube.wu.ac.at).

- **"custom"** custom user-provided extraction engine.

Control parameters for engine "xpdf" are as follows.

- **info**  
  a character vector specifying options passed over to the pdfinfo executable.

- **text**  
  a character vector specifying options passed over to the pdftotext executable.

Control parameters for engine "custom" are as follows.
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 Author (as character string), CreationDate (of class POSIXlt), Subject (as character string), Title (as character string), and Creator (as character string).

text a function extracting content from a PDF. The function must accept a file path as first argument and must return a character vector.

Value

A function with the following formals:

  elem a named list with the 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.

Examples

uri <- sprintf("file://%s", system.file(file.path("doc", "tm.pdf"), package = "tm"))
if(all(file.exists(Sys.which(c("pdfinfo", "pdftotext"))))) { 
  pdf <- readPDF(control = list(text = "-layout"))(elem = list(uri = uri),
                         language = "en",
                         id = "id1")
  content(pdf)[1:13]
}
VCorpus(URISource(uri, mode = ""),
        readerControl = list(reader = readPDF(engine = "ghostscript")))

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

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

Description

Read in a Reuters Corpus Volume 1 XML document.

Usage

```r
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

```r
f <- system.file("texts", "rcv1_2330.xml", package = "tm")
rcv1 <- readRCV1(elem = list(content = readLines(f)),
                 language = "en", id = "id1")
meta(rcv1)
```

---

**readReut21578XML**  
Read In a Reuters-21578 XML Document

Description

Read in a Reuters-21578 XML document.

Usage

```r
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.

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. [http://modnlp.berlios.de/reuters21578.html](http://modnlp.berlios.de/reuters21578.html)
readTabular

Read In a Text Document

Description

Return a function which reads in a text document from a tabular data structure (like a data frame or a list matrix) with knowledge about its internal structure and possible available metadata as specified by a so-called mapping.

Usage

readTabular(mapping)

Arguments

mapping  A named list of characters. The constructed reader will map each character entry to the content or metadata 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.

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 mapping) 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 a PlainTextDocument representing the text and metadata extracted from elem$content. The arguments language and id are used as fallback if no corresponding metadata entries are found in elem$content.

See Also

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

Vignette 'Extensions: How to Handle Custom File Formats'.
**readXML**  

**Examples**

```r
df <- data.frame(contents = c("content 1", "content 2", "content 3"),  
title = c("title 1", "title 2", "title 3") ,  
authors = c("author 1", "author 2", "author 3") ,  
topics = c("topic 1", "topic 2", "topic 3") ,  
stringsAsFactors = FALSE)
m <- list(content = "contents", heading = "title",  
          author = "authors", topic = "topics")
myReader <- readTabular(mapping = m)
ds <- DataFrameSource(df)
elem <- getElem(stepNext(ds))
(result <- myReader(elem, language = "en", id = "id1"))
meta(result)
```

---

**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 metadata 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 expression `spec` extracts information from an XML node.
  - `type = "attribute", spec = "XPathExpression"` The XPath expression `spec` extracts information from an attribute of an XML node.
  - `type = "function", spec = function(tree) ...` The function `spec` is called, passing over a tree representation (as delivered by `xmlInternalTreeParse` from package `XML`) of the read in XML document 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`.

Examples

```r
readGmane <-
readXML(spec = list(author = list("node", "/item/creator"),
content = list("node", "/item/description"),
datetimestamp = list("function", function(node)
strptime(sapply(XML:::getNodeset(node, "/item/date"), XML:::xmlValue),
  format = "%Y-%m-%dT%H:%M:%S",
  tz = "GMT")),
description = list("unevaluated", ""),
heading = list("node", "/item/title"),
id = list("node", "/item/link"),
origin = list("unevaluated", "Gmane Mailing List Archive"),
doc = PlainTextDocument())

removeNumbers(x, ...)
```

Description

Remove numbers from a text document.

Usage

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

**Arguments**

- `x` A text document.
- `...` Not used.

**Value**

The text document without numbers.

**See Also**

- `getTransformations` to list available transformation (mapping) functions.

**Examples**

```r
data("crude")
crude[[1]]
removeNumbers(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_dashes = FALSE)
## S3 method for class 'PlainTextDocument'
removePunctuation(x, ...)
```

**Arguments**

- `x` A character or text document.
- `preserve_intra_word_dashes` a logical specifying whether intra-word dashes should be kept.
- `...` passed over argument `preserve_intra_word_dashes`.

**Value**

The character or text document `x` without punctuation marks (besides intra-word dashes if `preserve_intra_word_dashes` is set).

**See Also**

- `getTransformations` to list available transformation (mapping) functions.
- `regex` shows the class `[:punct:]` of punctuation characters.
Example:

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

---

**removeSparseTerms**

*Remove Sparse Terms from a Term-Document Matrix*

**Description**

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

**Usage**

```r
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.

Example:

```r
data("crude")
```
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"))
```

---

Source  Sources

Description

Creating and accessing sources.

Usage

```r
SimpleSource(encoding = ",",
    length = 0,
    position = 0,
    reader = readPlain,
    ...,
    class)
getSources()
## S3 method for class 'SimpleSource'
eoi(x)
## S3 method for class 'DataframeSource'
getElem(x)
```
Arguments

x 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).
... tag-value pairs for storing additional information.
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 \texttt{R} vector, in order to acquire content in a uniform way. In packages which employ the infrastructure provided by package \texttt{tm}, such sources are represented via the virtual \texttt{S3} class \texttt{Source}: such packages then provide \texttt{S3} source classes extending the virtual base class (such as \texttt{DirSource} provided by package \texttt{tm} itself).

All extension classes must provide implementations for the functions \texttt{eoi}, \texttt{getElement}, \texttt{length}, \texttt{reader}, and \texttt{stepNext}. For parallel element access the function \texttt{pgetElement} must be provided as well.

The function \texttt{eoi} indicates end of input. \texttt{getElement} fetches the element at the current position, whereas \texttt{pgetElement} retrieves all elements in parallel at once. The function \texttt{length} gives the number of elements. \texttt{reader} returns a default reader for processing elements. \texttt{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`.
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

```r
text completion(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)

<table>
<thead>
<tr>
<th>stemDocument</th>
<th>Stem Words</th>
</tr>
</thead>
</table>

Description

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

Usage

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

Arguments

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

Details

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

Examples

data("crude")
crude[[1]]
stemDocument(crude[[1]])
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 (obtained from http://jmlr.csail.mit.edu/papers/volume5/lewis04a/11-smart-stop-list/english.stop) (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
```r
## S3 method for class 'PlainTextDocument'
stripWhitespace(x, ...)
```

Arguments
- `x`: A text document.
- `...`: Not used.

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

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

Examples
```r
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
```r
TermDocumentMatrix(x, control = list())
DocumentTermMatrix(x, control = list())
as.TermDocumentMatrix(x, ...)
as.DocumentTermMatrix(x, ...)
```
TermDocumentMatrix

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. 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", "texas"), c("127", "144", "191", "194")])
inspect(dtm[1:5, 273:276])
**termFreq**  
*Term Frequency Vector*

**Description**

Generate a term frequency vector from a text document.

**Usage**

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

**Arguments**

- **doc**: An object inheriting from `TextDocument`.
- **control**: A list of control options which override default settings. First, following two options are processed.

  - **tokenize**: A function tokenizing documents into single tokens or a string matching one of the predefined tokenization functions:  
    "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.

- **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 bounds$local{1}$ or more often than the upper bound bounds$local{2}$ are discarded. Defaults to list(local = c(1, Inf)) (i.e., every token will be used).

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

Value
A named integer vector of class term_frequency 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)
```

TextDocument

Description
Representing and computing on text documents.

Details

Text documents are documents containing (natural language) text. The tm package employs the infrastructure provided by package NLP and represents text documents via the virtual S3 class TextDocument. Actual S3 text document classes then extend the virtual base class (such as PlainTextDocument). All extension classes must provide an 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 content and meta methods for accessing the (possibly raw) document content and metadata.
See Also

`PlainTextDocument`, and `XMLTextDocument` for the text document classes provided by package `tm`.

`TextDocument` for text documents in package `NLP`.

---

**tm_combine**

Combine Corpora, Documents, Term-Document Matrices, and Term Frequency Vectors

---

**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

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 'VCorpus'
tm_filter(x, FUN, ...)
## S3 method for class 'PCorpus'
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 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

## S3 method for class 'PCorpus'
```r
tm_map(x, FUN, ...)
```

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

Arguments

- **x**  
  A corpus.

- **FUN**  
  a transformation function taking a text document as input and returning a text document. 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

```r
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_reduced

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, 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 = slam::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 = slam::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

tokenizer

Tokenizers

Description

Tokenize a document or character vector.

Usage

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.

scan_tokenizer Relies on scan(..., what = "character").
MC_tokenizer Implements the functionality of the tokenizer in the MC toolkit (http://www.cs.utexas.edu/users/dml/software/mc/).

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.

Examples

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

**Uniform Resource Identifier Source**

**Description**

Create a uniform resource identifier source.

**Usage**

```r
URISource(x, encoding = "", mode = "text")
```

**Arguments**

- `x`: A character vector of uniform resource identifiers (URIs).
- `encoding`: A character string describing the current encoding. It is passed to `iconv` to convert the input to UTF-8.
- `mode`: A character string specifying if and how URIs should be read in. Available modes are:
  - "" (No read. In this case `getElement` and `pGetElem` only deliver URIs).
  - "binary" (URIs are read in binary raw mode (via `readBin`).
  - "text" (URIs are read as text (via `readLines`).

**Details**

A uniform resource identifier source interprets each URI as a document.

**Value**

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

**See Also**

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

**Examples**

```r
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: For VCorpus a Source object, and for as.VCorpus an R 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").

Details
A volatile corpus is fully kept in memory and thus all changes only affect the corresponding R object.
The function Corpus is a convenience alias to VCorpus.

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), list(reader = readReut21578XMLasPlain))
**VectorSource**  
*Vector Source*

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

---

**weightBin**  
*Weight Binary*

**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.

---

**WeightFunction**

**Weighting Function**

Description

Construct a weighting function for term-document matrices.

Usage

\[ \text{WeightFunction}(x, \text{name}, \text{acronym}) \]

Arguments

\begin{itemize}
  \item \text{x} A function which takes a TermDocumentMatrix with term frequencies as input, weights the elements, and returns the weighted matrix.
  \item \text{name} A character naming the weighting function.
  \item \text{acronym} A character giving an acronym for the name of the weighting function.
\end{itemize}

Value

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

Examples

\begin{verbatim}
weightCutBin <- WeightFunction(function(m, cutoff) m > cutoff,
                                 "binary with cutoff", "bincut")
\end{verbatim}
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) \( tf_{i,j} \) 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(\( tf_{i,j} \)).

"a" (augmented) is defined as 0.5 + 0.5 * \( tf_{i,j} \max_{i}(tf_{i,j}) \).

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

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

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 \( \frac{N}{df_t} \) where \( df_t \) denotes how often term \( t \) occurs in all documents.

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

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 \( \text{slope} \ast \sqrt{\text{col sums}(m^2)} + (1 - \text{slope}) \ast \text{pivot} \) where both \( \text{slope} \) and \( \text{pivot} \) must be set via named tags in the control list.

"b" (byte size) is defined as \( \frac{1}{\text{CharLength}} \). 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

**Description**

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

**Usage**

```r
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

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

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, reader)
Arguments

x  a character giving a uniform resource identifier.
parser  a function accepting an XML tree (as delivered by xmlTreeParse in package XML) as input and returning a list of XML elements.
reader  a function capable of turning XML elements 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.

Examples

## An implementation for readGmane is provided as an example in ?readXML
eexample(readXML)

## Construct a source for a Gmane mailing list RSS feed.
GmaneSource <-
function(x,)
XMLSource(x,  
  function(tree) {
    nodes <- XML::xmlChildren(XML::xmlRoot(tree))  
    nodes[names(nodes) == "item"]  
  },  
  readGmane)
## Not run: gs <- GmaneSource("http://rss.gmane.org/gmane.comp.lang.r.general")
elem <- elem NextNext(gs)  
(gmane <- readGmane(elem, language = "en", id = "id1"))  
meta(gmane)
## End(Not run)
Usage

XMLTextDocument(x = list(),
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("exampleData", "test.xml", package = "XML")
(xtd <- XMLTextDocument(XML::xmlTreeParse(xml),
heading = "XML text document",
id = xml,
language = "en"))

meta(xtd)
as.character(xtd)
Explore Zipf's law and Heaps' law, two empirical laws in linguistics describing commonly observed characteristics of term frequency distributions in corpora.

**Usage**

```r
Zipf_plot(x, type = "1", ...)  
Heaps_plot(x, type = "1", ...)
```

**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.

**Details**

Zipf's law (e.g., [http://en.wikipedia.org/wiki/Zipf%27s_law](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](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)
```
Index

*Topic IO
  foreign, 10
*Topic datasets
  acq, 3
crude, 5
*Topic file
  readPDF, 20
  stopwords, 33
*Topic math
  termFreq, 36
  {, 5
[[, 5
acq, 3
as.character, 37
as.DocumentTermMatrix
  (TermDocumentMatrix), 34
as.list, 5
as.TermDocumentMatrix
  (TermDocumentMatrix), 34
as.VCorpus (VCorpus), 45
c.c.term_frequency (tm_combine), 38
c.c.TermDocumentMatrix (tm_combine), 38
c.c.TextDocument (tm_combine), 38
c.c.VCorpus (tm_combine), 38
content, 4, 37
c.content_transformer, 4, 12, 40
Corpus, 4, 13, 15, 31, 45
crude, 5
DataframeSource, 6, 31
DCorpus, 5
dir, 7
DirSource, 7, 30, 31
Docs, 8
document-term matrix, 11
DocumentTermMatrix, 8–10, 28
DocumentTermMatrix
  (TermDocumentMatrix), 34
DublinCore (meta), 13
DublinCore<-(meta), 13
Encoding, 7, 44
eoi (Source), 29
filehashOption, 15
findAssocs, 9
findFreqTerms, 9
foreign, 10
FunctionGenerator (Reader), 19
getElem, 7, 19, 44
getElem (Source), 29
getReaders (Reader), 19
getSources (Source), 29
getTokenizers, 11, 37, 43
getTransformations, 12, 27, 29, 34, 40, 41
graphNEL, 17
Heaps_plot (Zipf_n_Heaps), 54
iconv, 7, 44
inspect, 12
language, 15, 16, 45, 53
length, 5
length.SimpleSource (Source), 29
MC_tokenizer, 11, 36
MC_tokenizer (tokenizer), 43
meta, 5, 13, 14, 37
meta<-.PCorpus (meta), 13
meta<-.TextDocument (meta), 13
meta<-.VCorpus (meta), 13
nDocs (Docs), 8
nTerms (Docs), 8
parse_ISO_8601_datetime, 16, 53
PCorpus, 5, 14, 45
TermDocumentMatrix, 8–10, 28, 34, 38, 42, 46–50
termFreq, 35, 36, 38, 42
Terms (Docs), 8
TextDocument, 13, 16, 19, 37, 38, 52, 53
tm_combine, 38
tm_filter, 39
tm_index (tm_filter), 39
tm_map, 4, 12, 40
tm_reduce, 41
tm_term_score, 42
tokenize, 43
tokenizer, 43
tolower, 36
URISource, 31, 44
VCorpus, 3, 5, 15, 38, 45
VectorSource, 31, 46
weightBin, 35, 46
WeightFunction, 35, 47
weightSMART, 35, 47
weightTf, 35, 49
weightTfIdf, 35, 50
words, 36
wordStem, 32
writeCorpus, 51
XMLDocument, 53
XMLSource, 26, 31, 51
XMLTextDocument, 22, 23, 38, 52
xmlTreeParse, 52
Zipf_n_Heaps, 54
Zipf_plot (Zipf_n_Heaps), 54