Package 'chinese.misc'

May 9, 2020

Type Package

Title Miscellaneous Tools for Chinese Text Mining and More

Version 0.2.2

Date 2020-05-11

Maintainer Jiang Wu <textidea@sina.com>

Description Efforts are made to make Chinese text mining easier, faster, and robust to errors. Document term matrix can be generated by only one line of code; detecting encoding, segmenting and removing stop words are done automatically.
Some convenient tools are also supplied.

License GPL-3

URL https://github.com/githubwwwjjj/chinese.misc/blob/master/README.md

Depends R (>= 3.6.0)

Imports jiebaR, NLP, tm (>= 0.7), stringi, slam (>= 0.1-37), Matrix, purrr

Encoding UTF-8

LazyLoad true

LazyData true

RoxygenNote 7.1.0

NeedsCompilation no

Author Jiang Wu [aut, cre] (from Capital Normal University)

Repository CRAN

Date/Publication 2020-05-09 19:20:02 UTC

R topics documented:

`chinese.misc-package` *Miscellaneous Tools for Chinese Text Mining and More*

Description

This package aims to help accomplish the basic tasks of Chinese text mining in a more efficient way. The manual in Chinese is in https://github.com/githubwwwjjj/chinese.misc. Compared with other packages and functions, the package puts more weight on the following three points: (1) It helps save users' time. (2) It helps decrease errors (it tolerates and corrects input errors, if it can; and if it cannot, it gives meaningful error messages). (3) Although the functions in this package depend on **tm** and **stringi**, several steps and the values of arguments have been specially set to facilitate processing Chinese text. For example, `corp_or_dtm` creates corpus or document term matrix, users only need to input folder names or file names, and the function will automatically detect file encoding, segment terms, modify texts, remove stop words. `txt2csv` and `csv2txt` help

convert the format of texts and do some data cleaning. And there are some functions for object class assertion and coercion.

Author(s)

Jiang Wu

Examples

```
require(tm)
# Since no Chinese character is allowed, here we
# use English instead.
# Make a document term matrix in 1 step, few arguments have
# to be modified by the user.
x <- c(
  "Hello, what do you want to drink?",
  "drink a bottle of milk",
  "drink a cup of coffee",
  "drink some water",
  "hello, drink a cup of coffee")
dtm <- corp_or_dtm(x, from = "v", type = "dtm")
# Coerce list containing data frames and other lists
df <- data.frame(matrix(c(66, 77, NA, 99), nr = 2))
l <- list(a = 1:4, b = factor(c(10, 20, NA, 30)), c = c('x', 'y', NA, 'z'), d = df)
l2 <- list(l, l, cha = c('a', 'b', 'c'))
as.character2(l2)
```

---

as.character2 *An Enhanced Version of as.character*

Description

This function manages to coerce one or more objects into a character vector. Unlike `as.character`, this function can handle data frames, lists and recursive lists (lists of lists), even when there are factor objects inside data frames and lists. If there is any `NULL` object in a list, `as.character2` will coerce that element into `character(0)` rather than the character "NULL", which is what `as.character` does. When the object is of class matrix or data frame, the function will open it by column. The order of characters in result manages to keep accordance with that of the input object.

Usage

```
as.character2(...)
```

Arguments

`...`	one or more objects to be coerced.

Value

a character vector

Examples

```
as.character2(NULL, NULL)
# Try a list of NULLs
null_list <- list(a = NULL, b = NULL, c = NULL)
# Compare the different results of as.character
# and as.character2. In fact, we usually
# want the latter one.
as.character(null_list)
as.character2(null_list)
# Try a list with a data frame in it
df <- data.frame(matrix(c(66,77,NA,99), nrow = 2))
l <- list(a = 1:4, b = factor(c(10,20,NA, 30)), c = c('x', 'y', NA, 'z'), d = df)
as.character2(l)
# Try a list of lists
l2 <- list(l, l, cha = c('a', 'b', 'c'))
as.character2(l2)
```

---

as.numeric2 *An Enhanced Version of as.numeric*

Description

This function coerces objects into a numeric vector. There are several differences between this function and `as.numeric`. First, if `as.character2` fails to coerce (this is usually because there are characters in the input object), it will raise an error and stop rather than to give a warning. Second, it can handle data frame object, list, and recursive list. Third, it can coerce number-like factors exactly into what users see on the screen.

Usage

```
as.numeric2(...)
```

Arguments

...	one or more objects to be coerced.

Value

a numeric vector, or, if fails, an error will be raised.

Examples

```
# Try to coerce data frame
a <- c(55, 66, 77, 88, 66, 77, 88)
b <- factor(a)
df <- data.frame(a, b)
as.numeric2(df, a*2)
# Try a list
l <- list(a, a*2)
as.numeric2(l)
# Try a list of lists
l2 <- list(l, l)
as.numeric2(l2)
```

`corp_or_dtm` *Create Corpus or Document Term Matrix with 1 Line*

Description

This function allows you to input a vector of characters, or a mixture of files and folders, it will automatically detect file encodings, segment Chinese texts, do specified modification, remove stop words, and then generate corpus or dtm (tdm). Since **tm** does not support Chinese well, this function manages to solve some problems. See Details.

Usage

```
corp_or_dtm(
  ...,
  from = "dir",
  type = "corpus",
  enc = "auto",
  mycutter = DEFAULT_cutter,
  stop_word = NULL,
  stop_pattern = NULL,
  control = "auto",
  myfun1 = NULL,
  myfun2 = NULL,
  special = "",
  use_stri_replace_all = FALSE
)
```

Arguments

`...`	names of folders, files, or the mixture of the two kinds. It can also be a character vector of texts to be processed when setting `from` to "v", see below.
`from`	should be "dir" or "v". If your inputs are filenames, it should be "dir" (default), If the input is a character vector of texts, it should be "v". However, if it is set to "v", make sure each element is not identical to filename in your working

directory; and, if they are identical, the function will raise an error. To do this check is because if they are identical, `jiebaR::segment` will take the input as a file to read!

`type` what do you want for result. It is case insensitive, thus those start with "c" or "C" represent a corpus result; and those start with "d" or "D" for document term matrix, and those start with "t" or "T" for term document matrix. Input other than the above represents a corpus result. The default value is "corpus".

`enc` a length 1 character specifying encoding when reading files. If your files may have different encodings, or you do not know their encodings, set it to "auto" (default) to let the function auto-detect encoding for each file.

`mycutter` the jiebar cutter to segment text. A default cutter is used. See Details.

`stop_word` a character vector to specify stop words that should be removed. If it is `NULL`, nothing is removed. If it is "jiebar", "jiebaR" or "auto", the stop words used by **jiebaR** are used, see `make_stoplist`. Please note the default value is `NULL`. Texts are transformed to lower case before removing stop words, so your stop words only need to contain lower case characters.

`stop_pattern` vector of regular expressions. These patterns are similar to stop words. Terms that match the patterns will be removed. Note: the function will automatically adds "^" and "$" to the pattern, which means first, the pattern you provide should not contain these two; second, the matching is complete matching. That is to say, if a word is to be removed, it not just contains the pattern (which is to be checked by `grepl`, but the whole word match the pattern.

`control` a named list similar to that which is used by `DocumentTermMatrix` or `TermDocumentMatrix` to create dtm or tdm. But there are some significant differences. Most of the time you do not need to set this value because a default value is used. When you set the argument to `NULL`, it still points to this default value. See Details.

`myfun1` a function used to modify each text after being read by `scancn` and before being segmented.

`myfun2` a function used to modify each text after they are segmented.

`special` a length 1 character or regular expression to be passed to `dir_or_file` to specify what pattern should be met by filenames. The default is to read all files. See `dir_or_file`.

`use_stri_replace_all`
default is FALSE. If it is TRUE, `stringi::stri_replace_all` is used to delete stop words, which has a slightly higher speed. This is still experimental.

Details

Package **tm** sometimes tries to segment an already segmented Chinese Corpus and put together terms that should not be put together. The function is to deal with the problem. It calls `scancn` to read files and auto-detect file encodings, and calls `jiebaR::segment` to segment Chinese text, and finally calls `tm::Corpus` to generate corpus. When creating DTM/TDM, it partially depends on `tm::DocumentTermMatrix` and `tm::TermDocumentMatrix`, but also has some significant differences in setting control argument.

Users should provide their jiebar cutter by `mycutter`. Otherwise, the function uses `DEFAULT_cutter` which is created when the package is loaded. The `DEFAULT_cutter` is simply `worker(write = FALSE)`. See `jiebaR::worker`.

As long as you have not manually created another variable called "DEFAULT_cutter", you can directly use `jiebaR::new_user_word(DEFAULT_cutter...)` to add new words. By the way, whether you manually create an object called "DEFAULT_cutter", the original loaded DEFAULT_cutter which is used by default by functions in this package will not be removed by you. So, whenever you want to use this default value, you do not need to set `mycutter` and keep it as default.

The argument `control` is very similar to the argument used by `tm::DocumentTermMatrix`, but is quite different and will not be passed to it! The permitted elements are below:

- (1) wordLengths: length 2 positive integer vector. 0 and `inf` is not allowed. If you only want words of 4 to 10, then set it to c(4, 10). If you do not want to limit the ceiling value, just choose a large value, e.g., c(4, 100). In package tm (>= 0.7), 1 Chinese character is roughly of length 2 (but not always computed by multiplying 2), so if a Chinese words is of 4 characters, the min value of wordLengths is 8. But here in `corp_or_dtm`, word length is exactly the same as what you see on the screen. So, a Chinese word with 4 characters is of length 4 rather than 8.
- (2) dictionary: a character vetcor of the words which will appear in DTM/TDM when you do not want a full one. If none of the words in the dictionary appears in corpus, a blank DTM/TDM will be created. The vector should not contain `NA`, if it does, only non-NA elements will be kept. Make sure at least 1 element is not `NA`. Note: if both dictionary and wordLengths appear in your control list, wordLengths will be ignored.
- (3) bounds: an integer vector of length 2 which limits the term frequency of words. Only words whose total frequencies are in this range will appear in the DTM/TDM. 0 and `inf` is not allowed. Let a large enough value to indicate the unlimited ceiling.
- (4) have: an integer vector of length 2 which limits the time a word appears in the corpus. Suppose a word appears 3 times in the 1st article and 2 times in the 2nd article, and 0 in the 3rd, then its bounds value = 3 + 2 + 0 = 5; but its have value = 1 + 1 + 0 = 2.
- (5) weighting: a function to compute word weights. The default is to compute term frequency. But you can use other weighting functions, typically `tm::weightBin` or `tm::weightTfIdf`.
- (6) tokenizer: this value is temporarily deprecated and it cannot be modified by users.

By default, the argument `control` is set to "auto", "auto1", or `DEFAULT_control1`, which are the same. This control list is created when the package is loaded. It is simply `list(wordLengths = c(1,25))`, Alternatively, `DEFAULT_control2` (or "auto2") is also created when loading package, which sets word length to 2 to 25.

Value

a corpus, or document term matrix, or term document matrix.

Examples

```
x <- c(
  "Hello, what do you want to drink?",
  "drink a bottle of milk",
  "drink a cup of coffee",
  "drink some water")
# The simplest argument setting
dtm <- corp_or_dtm(x, from = "v", type = "dtm")
```

```
# Modify argument control to see what happens
dtm <- corp_or_dtm(x, from = "v", type="d", control = list(wordLengths = c(3, 20)))
tdm <- corp_or_dtm(x, from = "v", type = "T", stop_word = c("you", "to", "a", "of"))
```

CQUOTE *Paste Words or Numbers with Quatation Marks*

Description

This function simply pastes things together and message them on the screen. Suppose you use `x=readClipboard()` and get "a", "b", "c", then, `CQUOTE(x)` will message `c("a","b","c")` on the screen.

Usage

```
CQUOTE(x, quote = NULL, blank = TRUE)
```

Arguments

x	a character, numeric or factor vector.
quote	whether to add quotation marks. Default is NULL, the function adds quotation marks to characters, not numbers. However, you can set it to TRUE or FALSE.
blank	if x are characters, whether blank (i. e., "") is to be kept. Default is TRUE.

Examples

```
a=letters[1: 5]
CQUOTE(a) # c("a", "b", "c", "d", "e")
b=1: 5
CQUOTE(b) # c(1, 2, 3, 4, 5)
CQUOTE(b, quote=TRUE) # c("1", "2", "3", "4", "5")
```

---

create_ttm *Create Term-Term Matrix (Term-Cooccurrence Matrix)*

Description

This is a convenient function to create term-term matrix from document-term matrix, term-document matrix, or a matrix that represents one of the two. Sparse matrix is used to speed up computing. The output can be either a matrix or a sparse matrix.

Usage

```
create_ttm(x, type = "dtm", tomatrix = FALSE, checks = TRUE)
```

Arguments

x	an object of class DocumentTermMatrix or TermDocumentMatrix, or a matrix which has its rownames or colnames as terms.
type	if x is a matrix, this argument tells whether it is a DTM or a TDM; for the former, a character starting with "D/d", and for the latter, starting with "T/t".
tomatrix	should be logical, whether to output a matrix result. If TRUE, a matrix representing a TTM is returned. If FALSE (default), a list is returned: the first element is a sparse matrix created by package Matrix, with no words, the second element is a character vector of these words.
checks	if x is a matrix, whether to check its validity, that is, whether it is numeric, all values are 0 or positive, there is no NA.

Examples

```
x <- c(
  "Hello, what do you want to drink?",
  "drink a bottle of milk",
  "drink a cup of coffee",
  "drink some water")
dtm <- corp_or_dtm(x, from = "v", type = "dtm")
ttm1 <- create_ttm(dtm)
ttm2 <- create_ttm(dtm, tomatrix = TRUE)
tdm <- t(dtm)
ttm3 <- create_ttm(tdm)
ttm_sparse <- ttm3[[1]]
ttm_ordinary <- as.matrix(ttm_sparse)
colnames(ttm_ordinary) <- ttm3[[2]]
rownames(ttm_ordinary) <- ttm3[[2]]
# You can also use Matrix::writeMM(ttm_sparse, filename)
# to write it on your disk.
```

---

csv2txt *Write Texts in CSV into Many TXT/RTF Files*

Description

The function writes texts in a given .csv file into separated .txt/.rtf files with file names added.

Usage

```
csv2txt(
  csv,
  folder,
  which,
  header = TRUE,
  na_in_csv = c(NA, "", " ", "?", "NA", "999"),
  na_in_txt = " ",
```

```
  name_col = NULL,
  ext = "txt"
)
```

Arguments

csv	a .csv file. One of its columns contains texts to be written.
folder	a name of a folder that stores the .txt/.rtf files created by the function. The folder may already exist. If it does not exist, the function will try to create it recursively. If it cannot be created, an error will be raised. See dir.create. Note: a name that contains no punctuation is preferred.
which	a number: which column of the csv file contains texts.
header	should the .csv file be read with its first row as header? This argument is passed to read.csv. Default is TRUE.
na_in_csv	character vector indicating what content in the .csv file's cells should be taken as NA. The default values are "", " ", "?", "NA", "999"; and you can specify other values. But whatever you specify, the default values will always be taken as NA. If you do not provide a character vector, the default values are used.
na_in_txt	a length 1 character specifying what to write into a .txt file if a csv cell is NA. The default is " " (a space).
name_col	a length 1 number to indicate which column of your data should be taken as filenames. If it is NULL (default), a unique number will be given to each file, See Detail. If a cell is taken to be NA, it will be converted to ""; if it is too long, only the first 90 characters are used; one or more blanks and punctuations will be replaced by " " (a space).
ext	the extension of files to be written. Should be "txt", "rtf" or "". If it is not one of the three, it is set to "".

Details

In writing .txt/.rtf files, the function gives each file a unique number as part of its filename. The mechanism is as follows: suppose you have 1234 files, as this number has four digits, a series of numbers 0001, 0002,...0012,...0300,...1234 are assigned rather than 1, 2,...12,...300,...1234. There are several reasons to do this: first, if name_col is NULL, this procedure automatically assigns names. Second, the column you specify may have duplicate names. Third, even the column does not have duplicate names, the process the function modifies the names to make them valid may also produce duplicate names. Fourth, numbers with full digits make it easy to sort them in any software.

Value

nothing is returned and .txt/rtf files are written into the folder.

Examples

```
## Not run:
# First, we create a csv file
x1 <- file.path(find.package("base"), "CITATION")
x2 <- file.path(find.package("base"), "DESCRIPTION")
```

```
txt2csv(x1, x2, must_txt = FALSE, csv = "x1x2csv.csv")
# Now try to write files
wd <- getwd()
wd <- gsub("/$|\\\$", "", wd)
f <- paste(wd, "x1x2csv", sep="/")
csv2txt(csv = "x1x2csv.csv", folder = f, which = 3, ext = "")

## End(Not run)
```

DEFAULT_control1 *A Default Value for corp_or_dtm 1*

Description

In the previous version, this list object is by default used by corp_or_dtm. In this version, it is not the default value but it can still be used by the user. See details in corp_or_dtm.

Usage

```
DEFAULT_control1
```

Format

An object of class list of length 2.

Details

The object specifies word length from 1 to 25. The second element, a tokenizer, is temporally deprecated. Also, DEFAULT_control2 sets length from 2 to 25.

Examples

```
require(tm)
x <- c(
  "Hello, what do you want to drink?",
  "drink a bottle of milk",
  "drink a cup of coffee",
  "drink some water")
dtm <- corp_or_dtm(x, from = "v", type = "dtm", control = DEFAULT_control1)
```

DEFAULT_control2 *A Default Value for corp_or_dtm 2*

Description

The object specifies word length from 2 to 25. The second element, a tokenizer, is temporally deprecated. Also, DEFAULT_control1 sets length from 1 to 25.

Usage

```
DEFAULT_control2
```

Format

An object of class list of length 2.

Examples

```
require(tm)
x <- c(
  "Hello, what do you want to drink?",
  "drink a bottle of milk",
  "drink a cup of coffee",
  "drink some water")
dtm <- corp_or_dtm(x, from = "v", type = "dtm", control = DEFAULT_control2)
```

DEFAULT_cutter *A Default Cutter*

Description

This is simply a jiebar object created when the package is loaded. write is set to FALSE, so as to prevent segmented text from being automatically written into disk.

Usage

```
DEFAULT_cutter
```

Format

An object of class jiebar (inherits from segment, jieba) of length 11.

Examples

```
require(jiebaR)
x <- c("drink a bottle of milk",
  "drink a cup of coffee",
 "drink some water")
seg_file(x, from = "v")
seg_file(x, from = "v", mycutter = DEFAULT_cutter)
```

`dictionary_dtm` *Making DTM/TDM for Groups of Words*

Description

A dictionary has several groups of words. Sometimes what we want is not the term frequency of this or that single word, but rather the total sum of words that belong to the same group. Given a dictionary, this function can save you a lot of time because it sums up the frequencies of all groups of words and you do not need to do it manually.

Usage

```
dictionary_dtm(
  x,
  dictionary,
  type = "dtm",
  simple_sum = FALSE,
  return_dictionary = FALSE,
  checks = TRUE
)
```

Arguments

`x`	an object of class DocumentTermMatrix or TermDocumentMatrix created by `corp_or_dtm` or `tm::DocumentTermMatrix` or `tm::TermDocumentMatrix`. But it can also be a numeric matrix and you have to specify its type, see below.
`dictionary`	a dictionary telling the function how you group the words. It can be a list, matrix, data.frame or character vector. Please see details for how to set this argument.
`type`	if x is a matrix, you have to tell whether it represents a document term matrix or a term document matrix. Character starting with "D" or "d" for document term matrix, and that with "T" or "t" for term document matrix. The default is "dtm".
`simple_sum`	if it is `FALSE` (default), a DTM/TDM will be returned. If `TRUE`, you will not see the term frequency of each word in each text. Rather, a numeric vector is returned, each of its element represents the sum of the corresponding group of words in the corpus as a whole.
`return_dictionary`	if `TRUE`, a modified dictionary is returned, which only contains words that do exist in the DTM/TDM. The default is `FALSE`.

checks The default is TRUE. This will check whether x and dictionary is valid. For dictionary, if the input is not a list of characters, the function will manage to convert. You should not set this to FALSE unless you do believe that your input is OK.

Details

The argument dictionary can be set in different ways:

- (1) list: if it is a list, each element represents a group of words. The element should be a character vector; if it is not, the function will manage to convert. However, the length of the element should be > 0 and has to contain at least 1 non-NA word.
- (2) matrix or data.frame: each entry of the input should be character; if it is not, the function will manage to convert. At least one of the entries should not be NA. Each column (not row) represents a group of words.
- (3) character vector: it represents one group of words.
- (4) Note: you do not need to worry about two same words existing in the same group, because the function will only count one of them. Neither should you worry about that the words in a certain group do not really exist in the DTM/TDM, because the function will simply ignore those non-existent words. If none of the words of that group exists, the group will still appear in the final result, although the total frequencies of that group are all 0's. By setting return_dictionary = TRUE, you can see which words do exist.

Value

if return_dictionary = FALSE, an object of class DocumentTermMatrix or TermDocumentMatrix is returned; if TRUE, a list is returned, the 1st element is the DTM/TDM, and the 2nd element is a named list of words. However, if simple_sum = TRUE, the DTM/TDM in the above two situations will be replaced by a vector.

Examples

```
x <- c(
  "Hello, what do you want to drink and eat?",
  "drink a bottle of milk",
  "drink a cup of coffee",
  "drink some water",
  "eat a cake",
  "eat a piece of pizza"
)
dtm <- corp_or_dtm(x, from = "v", type = "dtm")
D1 <- list(
  aa <- c("drink", "eat"),
  bb <- c("cake", "pizza"),
  cc <- c("cup", "bottle")
)
y1 <- dictionary_dtm(dtm, D1, return_dictionary = TRUE)
#
# NA, duplicated words, non-existent words,
# non-character elements do not affect the
```

```
# result.
D2 <-list(
  has_na <- c("drink", "eat", NA),
  this_is_factor <- factor(c("cake", "pizza")),
  this_is_duplicated <- c("cup", "bottle", "cup", "bottle"),
  do_not_exist <- c("tiger", "dream")
)
y2 <- dictionary_dtm(dtm, D2, return_dictionary = TRUE)
#
# You can read into a data.frame
# dictionary from a csv file.
# Each column represents a group.
D3 <- data.frame(
  aa <- c("drink", "eat", NA, NA),
  bb <- c("cake", "pizza", NA, NA),
  cc <- c("cup", "bottle", NA, NA),
  dd <- c("do", "to", "of", "and")
)
y3 <- dictionary_dtm(dtm, D3, simple_sum = TRUE)
#
# If it is a matrix:
mt <- t(as.matrix(dtm))
y4 <- dictionary_dtm(mt, D3, type = "t", return_dictionary = TRUE)
```

`dir_or_file` *Collect Full Filenames from a Mix of Directories and Files*

Description

The input can be one or more directories, one or more files, or the mixture of the two. It will return the full paths of all files in a recursive way, and sort them in increasing order. When files are put in different areas of your disk, you may need this function to collect them. It is essentially a wrapper of `list.files`.

Usage

```
dir_or_file(..., special = "")
```

Arguments

`...`	names of directories and files; if the input is not vector, the function will try to coerce it. Relative paths and paths starting with "~/" are also accepted. In Windows, both "/" and double inversed slashes inside filenames are accepted.
`special`	a length 1 character or regular expression. Only filenames that have this pattern will be collected. Default value is "" (character with size 0), and is to collect everything.

Details

Failure may occur when obtaining absolute paths, please see `normalizePath` for possible reasons.

Value

a character vector of full filenames with increasing order, and every name is unique. If no filename is collected, an error will be raised.

Examples

```
x1 <- find.package("base")
x2 <- find.package("utils")
all_file <- dir_or_file(x1, x2, special = "rds$")
```

`get_tag_word` *Extract Words of Some Certain Tags through Pos-Tagging*

Description

Given a group of Chinese texts, this function manages to extract words of some specified types. For example, sometimes you want to collect all verbs that are used in your texts. Note: this function uses `jiebaR::tagging` to segment texts and do pos-tagging. The types assigned are not all correct. So, alternatively, you can first pos-tag your texts with other methods and then use this function.

Usage

```
get_tag_word(
  x,
  tag = NULL,
  tag_pattern = NULL,
  mycutter = DEFAULT_cutter,
  type = "word",
  each = TRUE,
  only_unique = FALSE,
  keep_name = FALSE,
  checks = TRUE
)
```

Arguments

`x`	it must be a list of character vectors, even when the list contains only one element. Each element of the list is either a length 1 character vector of a text, or a length >= 1 character vector which is the result of former tagging work. It should not contain `NA`.
`tag`	one or more tags should be specified. Words with these tags will be chosen. Possible tags are "v", "n", "vn", etc.
`tag_pattern`	should be a length 1 regular expression. You can specify tags by this pattern rather than directly provide tag names. For example, you can specify tag names starting with "n" by `tag_pattern = "^n"`. At least and at most one of tag and tag_pattern should be `NULL`.

mycutter	a cutter created with package jiebaR and given by users to tag texts. If your texts have already been pos-tagged, you can set this to NULL. By default, a DEFAULT_cutter is used, which is assigned as worker(write = FALSE) when loading the package.
type	if it is "word" (default), then extract the words that match your tags. If it is "position", only the positions of the words are returned. Note: if it is "positions", argument each (see below) will always be set to TRUE.
each	if this is TRUE (default), the return will be a list, each element of which is a extraction result of a text. If it is FALSE, the return will be a character vector with extracted words. See detail.
only_unique	if it is TRUE, only unique words are returned. The default is FALSE. See detail.
keep_name	whether to keep the tag names of the extracted words. The default is FALSE. Note: if only_unique = TRUE, all tag names will be removed.
checks	whether to check the correctness of arguments. The default is TRUE.

Details

The Argument each and only_unique decide what kind of return you can get.

- if each = TRUE and only_unique = FALSE, you can get a list, each element of which contains words extracted. This is the default.
- if each = TRUE and only_unique = TRUE, each element of the list only contains unique words.
- if each = FALSE and only_unique = FALSE, all words extracted will be put into a single vector.
- if each = FALSE and only_unique = TRUE, words extracted will be put into a single vector, but only unique words will be returned.

Examples

```
# No Chinese, so use English instead.
x1 <- c(v = "drink", xdrink = "coffee", v = "drink", xdrink = "cola", v = "eat", xfood = "banana")
x2 <- c(v = "drink", xdrink = "tea", v = "buy", x = "computer")
x <- list(x1, x2)
get_tag_word(x, tag = "v", mycutter = NULL)
get_tag_word(x, tag = "v", mycutter = NULL, only_unique = TRUE)
get_tag_word(x, tag_pattern = "^x", mycutter = NULL)
get_tag_word(x, tag_pattern = "^x", mycutter = NULL, keep_name = TRUE)
get_tag_word(x, tag = "v", mycutter = NULL, each = FALSE)
get_tag_word(x, tag = "v", mycutter = NULL, each = FALSE, only_unique = TRUE)
get_tag_word(x, tag = "v", mycutter = NULL, type = "position")
```

get_tmp_chi_locale *Check The Locale Functions are to Assume*

Description

The locale setting of R is different on different operating systems or different versions of one system. However, some functions in this package try to convert the locale setting of R to a new value. The new value, by default, is "Chinese (Simplified)_China.936" in Windows, and "zh_CN.UTF-8" in other systems. But users can modify this by `options(tmp_chi_locale = "...")` and then check this by `get_tmp_chi_locale()`. Note: if this value is `NULL` or `NA`, it means no locale modification will be done by functions in this package. If this value is "auto", it will be automatically converted to the default values.

Usage

```
get_tmp_chi_locale()
```

is_character_vector *A Convenient Version of is.character*

Description

This function checks to see if the object is a character vector. It is designed to have different actions from `is.character` and thus sometimes more convenient. See Details.

Usage

```
is_character_vector(x, len = NULL, allow_all_na = TRUE)
```

Arguments

`x`	object to be checked
`len`	numeric vector represents the permitted length of character vector. If an object is a character vector, but its length is not in `len`, the function still returns `FALSE`. The default is `NULL`, which means any length is OK.
`allow_all_na`	for length>1 character vector whose elements are all `NA`, if this argument is `FALSE`, then the function returns `FALSE`, if this argument is `TRUE` (default), then returns `TRUE`.

Details

Sometimes we want to check if an object is a character vector. But `is.character` cannot do this, because it also returns `TRUE` for a character matrix or data frame. What's more, we usually not only want to see if an object is of class character, but also want to see if it is valid, that is, can be passed to other functions without errors. But `is.character` even returns `TRUE` for `character(0)`. Also, `is.character(NA)` returns `FALSE`, but `is.character(as.character(NA))` returns `TRUE`, but in fact there is really no difference between the two for users and many functions that do not allow `NA`.

We list below the returns of `is.character2`:

- (1) if the object is `NULL`, `is.character2` returns `FALSE`.
- (2) if the object is of length 0, it always returns `FALSE`.
- (3) if the object is not vector, `FALSE`.
- (4) if it has only one element and this element is `NA`, under all circumstances it returns `FALSE`.
- (5) if the vector is of length>1, all the elements are `NA`, but the vector's class is not character, it returns `FALSE`.
- (6) if a character vector is of length>1, and all the elements are `NA`, then the result depends on argument `allow_all_na`, if `allow_all_na = TRUE`, then `TRUE`, otherwise, `FALSE`.

Value

`TRUE` or `FALSE`.

Examples

```
is_character_vector(character(0))
is_character_vector(NA)
is_character_vector(as.character(NA))
is_character_vector(c(NA, NA))
is_character_vector(as.character(c(NA,NA)))
is_character_vector(as.character(c(NA, NA)), allow_all_na = FALSE)
is_character_vector(as.character(c(NA, NA)), allow_all_na = TRUE)
is_character_vector(matrix(c("a", "b", "c", "d"), nr = 2))
is_character_vector(c("a", "b", "c"), len = c(1, 10))
is_character_vector(c("a", "b", "c"), len = c(1:10))
```

---

is_positive_integer *A Convenient Version of is.integer*

Description

This function checks if all elements of an object can be taken to be valid integers.

Usage

```
is_positive_integer(x, len = NULL)
```

Arguments

x	an object to be checked
len	numeric vector specifying the allowed length of the x. If the length of the checked object is not in len, the function will return FALSE, even when it is a positive integer vector. The default is NULL, which means any length is OK.

Details

The reasons to use is_positive_integer are:

- (1) We often check if an object is a vector of positive integer. But is.numeric cannot do this because it also returns TRUE for a numeric matrix.
- (2) Sometimes is.integer returns a too strict result. For example, is.integer(3.0) returns FALSE, but the number 3.0 is valid in codes such as rep(10,3.0), that is to say, as long as a number can be taken to be a valid integer, we take it to be a integer, even when is.integer returns FALSE.
- (3) is_positive_integer returns FALSE for length = 0 object, even when it is integer(0). To let the function return this result is because integer of length 0 is a invalid input for many functions.
- (4) is_positive_integer returns FALSE for any object that contains NA, so that object that gets a TRUE from this function is more likely to be a valid value to be passed to other functions.

Value

TRUE or FALSE

Examples

```
is_positive_integer(NULL)
is_positive_integer(as.integer(NA))
is_positive_integer(integer(0))
is_positive_integer(3.0)
is_positive_integer(3.3)
is_positive_integer(1:5)
is_positive_integer(1:5, len = c(2, 10))
is_positive_integer(1:5, len = c(2:10))
```

---

m2doc *Rewrite Terms and Frequencies into Many Files*

Description

Given a matrix representing a document term matrix, this function takes each row as term frequencies for one file, and rewrite each row as a text. Some text mining tools other than R accept segmented Chinese texts. If you already convert texts into a matrix, you can use this function to convert it into texts, corpus or create document term matrix again.

Usage

```
m2doc(m, checks = FALSE)
```

Arguments

m	a numeric matrix, data frame is not allowed. It must represent a document term matrix, rather than a term document matrix. Each row of the matrix represents a text. The matrix should have column names as terms to be written, but if it is NULL, the function will take them as "term1", "term2", "term3", ...No NA in the matrix is allowed.
checks	should be TRUE or FALSE. If it is TRUE, the function will check whether there is any NA in the input, whether it is numeric, and whether there is any negative number. Default is FALSE to save time.

Value

a character vector, each element is a text with repeated terms (by rep) linked by a space.

Examples

```
s <- sample(1:5, 20, replace = TRUE)
m <- matrix(s, nrow = 5)
colnames(m) <- c("r", "text", "mining", "data")
m2doc(m)
```

---

m3m *Convert Objects among matrix, dgCMatrix, simple_triplet_matrix, DocumentTermMatrix, TermDocumentMatrix*

Description

This is to convert objects conveniently. The three types of matrix are 1st, "matrix"; 2nd, "dgCMatrix" in package Matrix; 3rd, "simple_triplet_matrix", "DocumentTermMatrix", "TermDocumentMatrix" in package slam, tm. This function is to be used when you read a csv file and want it to be a dtm; or, when you have a very large dtm and you want it to be saved or passed to another function that deals with dgCMatrix object. Note, it cannot convert between simple_triplet_matrix on one side, and dtm or tdm on the other.

Usage

```
m3m(x, to, keep_name = TRUE)
```

Arguments

x	object of class matrix, dgCMatrix, simple_triplet_matrix, DocumentTermMatrix, TermDocumentMatrix.
to	to what class do you want to convert x to. Abbreviations can be used: "matrix" and "m" mean "matrix": "dgCMatrix" and "M" mean "dgCMatrix"; "simple_triplet_matrix" and "stm" mean "simple_triplet_matrix"; "DocumentTermMatrix", "dtm", "DTM" mean "DocumentTermMatrix"; "TermDocumentMatrix", "tdm", "TDM" mean "TermDocumentMatrix".
keep_name	whether to keep names or dimnames, which are, for dtm-like object, documents and terms. TRUE by default. If you set it to FALSE, you will lose them. But if you convert dgCMatrix to dtm or tdm, it is required that the dgCMatrix object has a list of length 2 as dimnames.

Value

the object whose class is specified by argument to.

Examples

```
# Make a matrix and convert to a dtm
m <- sample(0: 1, 50, replace = TRUE)
m <- matrix(m, nrow = 5)
colnames(m) <- letters[1: 10]
rownames(m) <- as.character(1: 5)
dtm <- m3m(m, "dtm")
# Convert dtm to dgCMatrix
M <- m3m(dtm, "M")
```

---

make_stoplist *Input a Filename and Return a Vector of Stop Words*

Description

When a filename is provided, the function will return a vector of terms. If nothing is provided, it will return the stop words used in package jiebaR. See Details.

Usage

```
make_stoplist(x = "jiebar", print = TRUE)
```

Arguments

x	a length 1 character specifying a valid stop word file. If it is not provided, or is "jiebar" (default), "jiebaR" or "auto", it will return part of the stop words used by package jiebaR. See Details.
print	TRUE or FALSE, whether to print the first 5 words

Details

In a valid text file that saves stop words, each word should occupy a single line. However, if any line that contains more than one word and these words are separated by blanks, punctuations, numbers, it is also accepted, for the function will try to split them. Duplicated words will also be automatically removed. The encoding of a stop words file is auto-detected by the function.

For stop word list from `jiebaR`, see `jiebaR::STOPPATH`. It contains many words that are often removed in analyzing Chinese text. However, the result returned by `make_stoplist` is slightly different.

Value

a character vector of words. If no word is obtained, it will return `NULL`.

`match_pattern` *Extract Strings by Regular Expression Quickly*

Description

Given a pattern and a character vector, the function will extract parts of these characters that match the pattern. It is simply a wrapper of `regmatches`.

Usage

```
match_pattern(pattern, where, vec_result = TRUE, perl = FALSE)
```

Arguments

`pattern`	a length 1 regular expression to be matched.
`where`	a character vector, each of its elements may or may not have parts that match the specified pattern.
`vec_result`	should be `TRUE` or `FALSE`. If `TRUE` (default), all matched parts will be returned in a character vector. If `FALSE`, a list is returned, each element of the list represents the matching result of the corresponding element in `where`. If an element in `where` has nothing matching the pattern, the result is still an element in the list and assigned `character(0)`.
`perl`	default is `FALSE`. Should Perl-compatible regexps be used?

Value

a character vector or a list. If an element in `where` is `NA`, the result corresponds to this element is `character(0)`.

Examples

```
p <- "x.*?y"
x <- c("x6yx8y", "x10yx30y", "aaaaaa", NA, "x00y")
y <- match_pattern(p, x)
y <- match_pattern(p, x, vec_result = FALSE)
```

output_dtm *Convert or Write DTM/TDM Object Quickly*

Description

Given a TermDocumentMatrix or DocumentTermMatrix object, the function converts it to a matrix or write it into a .csv file, with additional filenames attached to it.

Usage

```
output_dtm(x, outputfile = NULL, doc_name = NULL)
```

Arguments

x	an object created by `tm::TermDocumentMatrix` or `tm::DocumentTermMatrix`.
outputfile	when it is NULL (default), no file is written and a matrix is returned. When a filename is provided, it will write the matrix into a file. The filename must end with ".csv".
doc_name	whether `NULL` or a character vector specifying the names you want to give to texts. If it is not a character vector, the function will try to coerce. Then the names become the row names of the returned matrix. Double inversed slashes will be converted to "/" by the function. The length of the argument must be equal to the number of files. `NA` element is not allowed. By default it is `NULL`, which means no name is added.

Examples

```
require(tm)
x <- c(
  "Hello, what do you want to drink?",
  "drink a bottle of milk",
  "drink a cup of coffee",
  "drink some water")
dtm <- corp_or_dtm(x, from = "v", type = "dtm")
output_dtm(dtm, doc_name = paste("doc", 1:4))
```

scancn *Read a Text File by Auto-Detecting Encoding*

Description

The function reads a text file and tries to detect file encoding. If you have Chinese files from different sources and cannot give them a single encoding, just let this function detect and read them. The function can save you much time on dealing with unrecognizable characters.

Usage

```
scancn(x, enc = "auto", read_2nd = TRUE, collapse = "   ")
```

Arguments

x	a length 1 character specifying filename.
enc	a length 1 character of file encoding specified by user. The default is "auto", which means let the function detect encoding.
read_2nd	should be TRUE (default) or FALSE. When it is TRUE, some files will be read twice, see Details.
collapse	this is used by the collapse argument of paste in order to link characters together. Default is " " (three spaces).

Details

The function calls `scan(x,what = "character",...)` and auto-detects file encoding. Sometimes a Chinese file is encoded in "UTF-8", but what is actually read is a "?". When this happens, the function reads it twice and uses `stringi::stri_encode` to convert it. If invalid inputs are found in the content, the file will also be read twice.

The function always returns a length 1 character. If the return of scan is a vector with length larger than 1, elements will be pasted together with three spaces or other specified symbols.

It will return a " " (one space) when all the elements of the vector are NA. If not all elements are NA, those equal to NA will be changed to "" (a size 0 string) before being pasted together.

Value

a length 1 character of text.

Examples

```
# No Chinese is allowed, so try an English file
x <- file.path(find.package("base"), "CITATION")
scancn(x)
```

---

seg_file *Convenient Tool to Segment Chinese Texts*

Description

The function first collects filenames or text vectors, then it calls `jiebaR::segment` to segment texts. In this process, it allows users to do additional modification. File encoding is detected automatically. After segmenting, segmented words that belong to a text will be pasted together into a single character with words split by " ". The segmented result will be returned or written on the disk.

Usage

```
seg_file(
  ...,
  from = "dir",
  folder = NULL,
  mycutter = DEFAULT_cutter,
  enc = "auto",
  myfun1 = NULL,
  myfun2 = NULL,
  special = "",
  ext = "txt"
)
```

Arguments

...	names of folders, files, or the mixture of the two kinds. It can also be a character vector of text to be processed when setting from to "v", see below.
from	should only be "dir" or "v". If your inputs are filenames, it should be "dir" (default), If the inputs is a character vector of texts, it should be "v". However, if it is set to "v", make sure each element of the vector is not identical to filename in your working directory; if they are identical, an error will be raised. To do this check is because if they are identical, the function segment will take the input as a file to read!
folder	a length 1 character indicating the folder to put the segmented text. Set it to NULL if you want the result to be a character vector rather than to be written on your disk. Otherwise, it should be a valid directory path, each segmented text will be written into a .txt/.rtf file. If the specified folder does not exist, the function will try to create it.
mycutter	the jiebar cutter to segment text. A default cutter is used. See Details.
enc	the file encoding used to read files. If files have different encodings or you do not know their encodings, set it to "auto" (default) to let encodings be detected automatically.
myfun1	a function used to modify each text after being read by scancn and before being segmented.
myfun2	a function used to modify each text after they are segmented.
special	a length 1 character or regular expression to be passed to dir_or_file to specify what pattern should be met by filenames. The default is to read all files.
ext	the extension of written files. Should be "txt", "rtf" or "". If it is not one of the three, it is set to "". This is only used when your input is a text vector rather than filenames and you want to write the outcome into your disk.

Details

Users should provide their jiebar cutter by mycutter. Otherwise, the function uses DEFAULT_cutter which is created when the package is loaded. The DEFAULT_cutter is simply worker(write = FALSE). See jiebaR::worker.

As long as you have not manually created another variable called "DEFAULT_cutter", you can directly use `jiebaR::new_user_word(DEFAULT_cutter...)` to add new words. By the way, whether you manually create an object called "DEFAULT_cutter", the original loaded DEFAULT_cutter which is used by default by functions in this package will not be removed by you. So, whenever you want to use this default value, either you do not set `mycutter`, or set it to `mycutter = chinese.misc::DEFAULT_cutter`.

The encoding for writing files (if `folder` is not NULL) is always "UTF-8".

Value

a character vector, each element is a segmented text, with words split by " ". If `folder` is a folder name, the result will be written into your disk and nothing returns.

Examples

```
require(jiebaR)
# No Chinese word is allowed, so we use English here.
x <- c("drink a bottle of milk",
  "drink a cup of coffee",
 "DRINK SOME WATER")
seg_file(x, from = "v", myfun1 = tolower)
```

---

slim_text *Remove Words through Speech Tagging*

Description

The function calls `jiebaR::tagging` to do speech tagging on a Chinese text, and then removes words that have certain tags.

Usage

```
slim_text(
  x,
  mycutter = DEFAULT_cutter,
  rm_place = TRUE,
  rm_time = TRUE,
  rm_eng = FALSE,
  rm_alpha = FALSE,
  paste = TRUE
)
```

Arguments

`x`	a length 1 character of Chinese text to be tagged
`mycutter`	a jiebar cutter provided by users to tag text. It has a default value, see Details.

rm_place	TRUE or FALSE. if TRUE (default), words related to a specified place ("ns") are removed.
rm_time	TRUE or FALSE. if TRUE (default), time related words ("t") are removed.
rm_eng	TRUE or FALSE. if TRUE, English words are removed. The default is FALSE.
rm_alpha	should be "any", TRUE or FALSE (default). Some English words are tagged as "x", so cannot be remove by setting rm_eng. But when rm_alpha is TRUE, any word that contains only a-zA-Z will be removed. If it is "any", then words that are mixtures of a-zA-Z and Chinese/digits will be removed.
paste	TRUE or FALSE, whether to paste the segmented words together into a length 1 character. The default is TRUE.

Details

Stop words are often removed from texts. But a stop word list hardly includes all words that need to be removed. So, before removing stop words, we can remove a lot of insignificant words by tagging and make the texts "slim". The webpage http://www.docin.com/p-341417726.html?_t_t_t=0.3930890985844252 provides details about Chinese word tags.

Only words with the following tags are to be preserved:

- (1) "n": nouns;
- (2) "t": time related words;
- (3) "s": space related words;
- (4) "v": verbs;
- (5) "a": adjectives;
- (6) "b": words only used as attributes in Chinese;
- (7) "x": strings;
- (8) "j", "l", "i", "z": some specific Chinese letters and phrases;
- (9) "unknown": words of unknown type;
- (10) "eng": English words.

Optionally, words related to a specified place ("ns"), time related words ("t") and english words ("eng") can be removed.

By default, a DEFAULT_cutter is used by the mycutter argument, which is assigned as worker(write = FALSE) when loading the package. As long as you have not manually created another variable called "DEFAULT_cutter", you can directly use jiebaR::new_user_word(DEFAULT_cutter...) to add new words. By the way, whether you manually create an object called "DEFAULT_cutter", the original loaded DEFAULT_cutter which is used by default by functions in this package will not be removed by you. So, whenever you want to use this default value, you just do not set mycutter.

Value

a length 1 character of segmented text, or a character vector, each element of which is a word.

Examples

```
require(jiebaR)
cutter <- jiebaR::worker()
# Give some English words a new tag.
new_user_word(cutter, c("aaa", "bbb", "ccc"),  rep("x", 3))
x <- "we have new words: aaa, bbb, ccc."
# The default is to keep English words.
slim_text(x, mycutter = cutter)
# Remove words tagged as "eng" but others are kept.
slim_text(x, mycutter = cutter, rm_eng = TRUE)
# Remove any word that only has a-zA-Z,
# even when rm_eng = FALSE.
slim_text(x, mycutter = cutter, rm_eng = TRUE, rm_alpha = TRUE)
slim_text(x, mycutter = cutter, rm_eng = FALSE, rm_alpha = TRUE)
```

---

sort_tf *Find High Frequency Terms*

Description

By inputting a matrix, or a document term matrix, or term document matrix, this function counts the sum of each term and output top n terms. The result can be messaged on the screen, so that you can manually copy them to other places (e. g., Excel).

Usage

```
sort_tf(x, top = 10, type = "dtm", todf = FALSE, must_exact = FALSE)
```

Arguments

x	a matrix, or an object created by corp_or_dtm or by tm::DocumentTermMatrix, or tm::TermDocumentMatrix. Data frame is not allowed. If it is a matrix, the column names (if type is "dtm") or row names (if type is "tdm") is taken to be terms, see below. If the names are NULL, terms are set to "term1", "term2", "term3"...automatically.
top	a length 1 integer. As terms are in the decreasing order of the term frequency, this argument decides how many top terms should be returned. The default is 10. If the number of terms is smaller than top, all terms are returned. Sometimes the returned terms are more than top, see below.
type	should start with "D/d" representing document term matrix, or "T/t" representing term document matrix. It is only used when x is a matrix. The default is "dtm".
todf	should be TRUE or FALSE. If it is FALSE (default) terms and their frequencies will be pasted by "&" and messaged on the screen, nothing is returned. Otherwise, terms and frequencies will be returned as data frame.
must_exact	should be TRUE or FALSE (default). It decides whether the number of returned words should be equal to that specified by top. See Details.

Details

Sometimes you may pick more terms than specified by `top`. For example, you specify to pick up the top 5 terms, and the frequency of the 5th term is 20. But in fact there are two more terms that have frequency of 20. As a result, `sort_tf` may pick up 7 terms. If you want the number is exactly 5, set `must_exact` to `TRUE`.

Value

return nothing and message the result, or return a data frame.

Examples

```
require(tm)
x <- c(
  "Hello, what do you want to drink?",
  "drink a bottle of milk",
  "drink a cup of coffee",
  "drink some water",
  "hello, drink a cup of coffee")
dtm <- corp_or_dtm(x, from = "v", type = "dtm")
# Argument top is 5, but more than 5 terms are returned
sort_tf(dtm, top = 5)
# Set must_exact to TRUE, return exactly 5 terms
sort_tf(dtm, top=5, must_exact=TRUE)
# Input is a matrix and terms are not specified
m=as.matrix(dtm)
colnames(m)=NULL
mt=t(m)
sort_tf(mt, top=5, type="tdm")
```

---

sparse_left *Check How many Words are Left under Certain Sparse Values*

Description

This function does not really remove sparse words (which is what `tm::removeSparseTerms` does); rather, it only shows how many words are left when you specify some sparse values. See Examples.

Usage

```
sparse_left(x, sparse)
```

Arguments

`x`	a DocumentTermMatrix or TermDocumentMatrix object.
`sparse`	a numeric vector with elements >= 0 and <= 1.

Examples

```
x <- c(
  "Hello, what do you want to drink?",
  "drink a bottle of milk",
  "drink a cup of coffee",
  "drink some water")
dtm <- corp_or_dtm(x, from = "v", type = "dtm")
y <- sparse_left(dtm, seq(0, 1, 0.1))
# Then you can use plot(sort(y, decreasing = TRUE), type = "b") to
# see which sparse value is proper.
```

---

tf2doc *Transform Terms and Frequencies into a Text*

Description

This function is simply a wrapper of `rep`, but allows different structures of input. For rewriting more texts in the same time, see `m2doc`.

Usage

```
tf2doc(term, num)
```

Arguments

`term`	terms that you want to rewrite into a text. A character vector is preferred, but matrix, list, data frame are also OK. `NA` in the argument will be taken as letters "NA" and repeated.
`num`	frequencies of terms in `term`. A numeric vector is preferred, but matrix, list, data frame are also OK. Its length must be equal to that of `term`. No `NA` is allowed.

Value

a character vector. Terms are pasted with a space.

Examples

```
x <- matrix(c("coffee", "milk", "tea", "cola"), nrow = 2)
y <- factor(c(5:8))
tf2doc(x, y)
```

topic_trend *Simple Rise or Fall Trend of Several Years*

Description

When topic names and corresponding years are given, this function computes the rise and fall trend during the period by `lm`.

Usage

```
topic_trend(year, topic, relative = FALSE, zero = 0)
```

Arguments

`year`	a numeric vector of years for corresponding topics, if it is not numeric, the function will try to coerce. The years should be written in full-digit, that is, if they are 1998 and 2013, do not simply write 98 and 13. No `NA` is allowed. And, the number of unique years is at least 3, otherwise an error will be raised.
`topic`	a character vector of topics. If it is not character, the function will try to coerce. The length of topic and year should be the same. No `NA` is allowed.
`relative`	if `FALSE` (default), the numbers of topics is used. If `TRUE`, the percentage of a topic in a year against the total number of that year is used. Suppose this year we have 200 texts on art, and the total number of texts in this year is 1000, then the relative value is 200/1000 = 0.2 rather than the absolute number 200. Note: if to use relative value, `NA` of the amount of a topic will be automatically set to 0.
`zero`	this can only be 0 (default) or `NA`. Suppose we have 0 text on a certain topic, then you will make sure whether the amount is really 0, or the data of this topic in that year is missing. Set this argument to `NA` to make all 0 into `NA`.

Details

The detail of trend info in the result is as follows:

- (1) trendIndex: a regression with function `lm` is done for every topic with year as x and amount of topics as y. The value of trendIndex is the slope k in y = kx+b.
- (2) trendLevel: the p value of k.
- (3) totalTrend: if trendIndex is larger than 0, then "rise", otherwise "fall". If trendLevel is smaller than 0.05, than "significant rise" or "significant fall".
- (4) maxminYear: if totalTrend is "rise" or "significant rise", then this value points out which year has the largest amount. If several years have the largest value, the most recent year is returned. If totalTrend is "fall" or "significant fall", the year has the smallest amount is returned.

- (5) detailTrend: if totalTrend is "rise" or "significant rise", then the function will see whether the year has the largest amount is the last year, if it is, then "rise along", otherwise "rise and fall". If totalTrend is "fall" or "significant fall", the function will see whether the year has the smallest amount is the last year, if it is, then "fall along", otherwise "fall and rise".
- (6) simpleTrend: it is simply whether the amount of the last year is larger than that of the first year. If yes, then "rise", if smaller, then "fall", if the same, then "equal".

When computing trend for a topic, if less than 3 years has valid value and value in other years are all NA, then trendIndex, trendLevel and maxminYear will be -999, and other cells are "less than 3y". If the numbers of a topic do not change through years, then trendIndex will be 0, trendLevel and maxminYear will be -999, totalTrend and detailTrend will be "almost same".

Value

a list. The 1st element is trend info. The 2nd is a summary of amount of each topic in each year. If argument relative is TRUE, a 3rd element is returned, which is the relative value (percentage) of each topic in each year.

Examples

```
set.seed(1)
topic <- sample(c("art", "economy", "law", "politics", "sociology"), 50, replace = TRUE)
set.seed(2)
year <- sample(2011: 2016, 50, replace = TRUE)
tr1 <- topic_trend(year, topic)
tr2 <- topic_trend(year, topic, zero = NA)
tr3 <- topic_trend(year, topic, relative=TRUE)
```

---

txt2csv *Write Many Separated Files into a CSV*

Description

Given filenames, folder names, or the mixture of the two, the function will read texts in .txt or other separated files, and then write them into one .csv file. It helps those who prefer texts in a table format.

Usage

```
txt2csv(..., csv, must_txt = TRUE, na_in_txt = NULL)
```

Arguments

...	names of folders and files, obtained files may end with ".txt" or not , see below. Encoding for each file is auto-detected.
csv	a .csv file that will contain texts. It must end with ".csv".
must_txt	should be TRUE or FALSE. Should all qualified texts end with ".txt"? If you want to read other types of file, such as .rtf, set it to FALSE. Default is TRUE.

na_in_txt	character vector that specifies what content, when it occupies a single line, should be treated as NA. See Details. Length of it can be larger than 1.

Details

Whether a file is taken as NA is judged by scancn. " " (a space) is also taken as NA. However, you can further decide what else is deemed as NA, e. g., "404 ERROR", if your texts are from websites. If a file cannot be accessed, the result to be written in the corresponding cell of csv file will become NA, and there will be a message, but no error is raised. In the .csv file, full filenames of txt occupy a column and fulltexts occupy another.

Examples

```
## Not run:
x1 <- file.path(find.package("base"), "CITATION")
x2 <- file.path(find.package("base"), "DESCRIPTION")
txt2csv(x1, x2, must_txt = FALSE, csv = 'x1x2csv.csv')

## End(Not run)
```

V *Copy and Paste from Excel-Like Files*

Description

These functions make it easy for copy and paste data from Excel-like files, especially when there are blank cells or when different columns have different lengths. All of them have the same arguments.

- V, when you do not copy rownames or colnames
- VR, when the 1st column is for rownames and there are no colnames in what you copy
- VC, when there are colnames but no rownames
- VRC and the same: VCR, when there are both rownames and colnames

If you copy something from a text document (e.g., Windows Notepad), the function may warn "incomplete final line found by readTableHeader...". This is because your content does not end with an end of line sign. You can simply ignore this warning!

Usage

```
V(tofactor = 0, keepblank = 0, sep = "\t")
```

Arguments

tofactor	if this is equal to numeric 1 or TRUE, characters will be converted to factors. Otherwise no conversion will be done. The default is not to convert.
keepblank	if characters are not to be converted to factors, this argument decides how to deal with blank cells in character columns. If it is numeric 1 or TRUE, a blank cell will be converted to "" (size 0 string). Otherwise it is viewed as NA (default).

sep	a single character to differentiate cells of a table. The default value should be used when your data is from Excel.

VC *Copy and Paste from Excel-Like Files*

Description

See V.

Usage

```
VC(tofactor = 0, keepblank = 0, sep = "\t")
```

Arguments

tofactor	if this is equal to numeric 1 or TRUE, characters will be converted to factors. Otherwise no conversion will be done. The default is not to convert.
keepblank	if characters are not to be converted to factors, this argument decides how to deal with blank cells in character columns. If it is numeric 1 or TRUE, a blank cell will be converted to "" (size 0 string). Otherwise it is viewed as NA (default).
sep	a single character to differentiate cells of a table. The default value should be used when your data is from Excel.

VCR *Copy and Paste from Excel-Like Files*

Description

See V.

Usage

```
VCR(tofactor = 0, keepblank = 0, sep = "\t")
```

Arguments

tofactor	if this is equal to numeric 1 or TRUE, characters will be converted to factors. Otherwise no conversion will be done. The default is not to convert.
keepblank	if characters are not to be converted to factors, this argument decides how to deal with blank cells in character columns. If it is numeric 1 or TRUE, a blank cell will be converted to "" (size 0 string). Otherwise it is viewed as NA (default).
sep	a single character to differentiate cells of a table. The default value should be used when your data is from Excel.

VR *Copy and Paste from Excel-Like Files*

Description

See V.

Usage

```
VR(tofactor = 0, keepblank = 0, sep = "\t")
```

Arguments

tofactor	if this is equal to numeric 1 or TRUE, characters will be converted to factors. Otherwise no conversion will be done. The default is not to convert.
keepblank	if characters are not to be converted to factors, this argument decides how to deal with blank cells in character columns. If it is numeric 1 or TRUE, a blank cell will be converted to "" (size 0 string). Otherwise it is viewed as NA (default).
sep	a single character to differentiate cells of a table. The default value should be used when your data is from Excel.

VRC *Copy and Paste from Excel-Like Files*

Description

See V.

Usage

```
VRC(tofactor = 0, keepblank = 0, sep = "\t")
```

Arguments

tofactor	if this is equal to numeric 1 or TRUE, characters will be converted to factors. Otherwise no conversion will be done. The default is not to convert.
keepblank	if characters are not to be converted to factors, this argument decides how to deal with blank cells in character columns. If it is numeric 1 or TRUE, a blank cell will be converted to "" (size 0 string). Otherwise it is viewed as NA (default).
sep	a single character to differentiate cells of a table. The default value should be used when your data is from Excel.

word_cor *Word Correlation in DTM/TDM*

Description

Given a DTM/TDM/matrix, the function computes the pearson/spearman/kendall correlation between pairs of words and filters the values by p value and minimum value of correlation. It is a little more flexible than `tm::findAssocs`.

Usage

```
word_cor(x, word, type = "dtm", method = "kendall", p = NULL, min = NULL)
```

Arguments

x	a DocumentTermMatrix, TermDocumentMatrix object, or a matrix. If it is a matrix, you must specify its type by the argument `type`. If it is a matrix, `NA` is not allowed, and rownames/colnames that are taken as words should not be `NULL`.
word	a character vector of words that you want to know their correlation in you data. If it is not a vector, the function will try to coerce. The length of it should not larger than 200. The function only computes for words that do exist in data, and those not in data will not be included.
type	if it starts with "d/D", it represents a DTM; if with "t/T", TDM; others are not valid. This is only used when x is a matrix. The default is "dtm".
method	what index is to be computed? It can only be "pearson", "spearman", or "kendall" (default). The method is passed to `stats::cor.test`. The default is "kendall".
p	if the p value of a correlation index is >= this value, the index will be convert to `NA` in the correlation matrix. The default is `NULL`, which means no filter is done. Note: if both argument p and min are non-Null, their relation is "or" rather than "and".
min	if the correlation index is smaller than this value, it will be convert to `NA`. The default is `NULL`, which means no filter is done.

Value

a list. The 1st element is the correlation matrix with diagonal converted to `NA`. The 2nd element is the p value matrix with diagonal converted to `NA`.

Examples

```
set.seed(1)
s <- sample(1:10, 100, replace = TRUE)
m <- matrix(s, nrow = 20)
myword<- c("alpha", "apple", "cake", "data", "r")
colnames(m) <- myword
```

```
mycor1 <- word_cor(m, myword)
mycor2 <- word_cor(m, myword, method = "pearson", min = 0.1, p = 0.4)
mt <- t(m)
mycor3 <- word_cor(mt, myword, type = "T", method = "spearman", p = 0.5)
```

Index