Package 'textTinyR'

June 5, 2017

Type Package

Title Text Processing for Small or Big Data Files

Version 1.0.7

Date 2017-06-05

Author Lampros Mouselimis <mouselimislampros@gmail.com>

Maintainer Lampros Mouselimis <mouselimislampros@gmail.com>

BugReports https://github.com/mlampros/textTinyR/issues

URL https://github.com/mlampros/textTinyR

Description Processes big text data files in batches efficiently. For this purpose, it offers functions for splitting, parsing, tokenizing and creating a vocabulary. Moreover, it includes functions for building either a document-term matrix or a term-document matrix and extracting information from those (term-associations, most frequent terms). Lastly, it embodies functions for calculating token statistics (collocations, look-up tables, string dissimilarities) and functions to work with sparse matrices. The source code is based on 'C++11' and exported in R through the 'Rcpp', 'RcppArmadillo' and 'BH' packages.

License GPL-3

Copyright inst/COPYRIGHTS

SystemRequirements The package requires the following two components : A C++11 compiler and on a unix OS the boost-locale headers and libraries (boost >= 1.55.0 , www.boost.org). Debian/Ubuntu: libboost-locale-dev, Fedora : yum install boost-devel, OSX/brew : detailed installation instructions can be found in the README file

LazyData TRUE

Depends R(>= 3.2.3), Matrix

Imports Rcpp (>= 0.12.5), R6, data.table, utils

LinkingTo Rcpp, RcppArmadillo (>= 0.7.5), BH

Suggests testthat, covr, knitr, rmarkdown

VignetteBuilder knitr

RoxygenNote 5.0.1

NeedsCompilation yes

Repository CRAN

Date/Publication 2017-06-05 15:13:23 UTC

R topics documented:

`big_tokenize_transform` *String tokenization and transformation for big data sets*

Description

String tokenization and transformation for big data sets

Usage

```
# utl <- big_tokenize_transform$new(verbose = FALSE)
```

Arguments

start_query	a character string. The *start_query* is the first word of the subset of the data and should appear frequently at the beginning of each line int the text file.
end_query	a character string. The *end_query* is the last word of the subset of the data and should appear frequently at the end of each line in the text file.
min_lines	a numeric value specifying the minimum number of lines. For instance if min_lines = 2, then only subsets of text with more than 1 lines will be kept.
trimmed_line	either TRUE or FALSE. If FALSE then each line of the text file will be trimmed both sides before applying the start_query and end_query
verbose	either TRUE or FALSE. If TRUE then information will be printed in the console
input_path_folder	a character string specifying the folder where the input files are saved
output_path_folder	a character string specifying the folder where the output files should be saved
input_path_file	a character string specifying the path to the input file
batches	a numeric value specifying the number of batches to use. The batches will be used to split the initial data into subsets. Those subsets will be either saved in files (*big_text_splitter* function) or will be used internally for low memory processing (*big_text_tokenizer* function).
read_file_delimiter	the delimiter to use when the input file will be red (for instance a tab-delimiter or a new-line delimiter).
to_lower	either TRUE or FALSE. If TRUE the character string will be converted to lower case
remove_char	a character string with specific characters that should be removed from the text file. If the *remove_char* is "" then no removal of characters take place
to_upper	either TRUE or FALSE. If TRUE the character string will be converted to upper case
utf_locale	the language specific locale to use in case that either the *to_lower* or the *to_upper* parameter is TRUE and the text file language is other than english. For instance if the language of a text file is greek then the *utf_locale* parameter should be *'el_GR.UTF-8'* (*language_country.encoding*). A wrong utf-locale does not raise an error, however the runtime of the function increases.
remove_punctuation_string	either TRUE or FALSE. If TRUE then the punctuation of the character string will be removed (applies before the split function)
remove_punctuation_vector	either TRUE or FALSE. If TRUE then the punctuation of the vector of the character strings will be removed (after the string split has taken place)
remove_numbers	either TRUE or FALSE. If TRUE then any numbers in the character string will be removed
trim_token	either TRUE or FALSE. If TRUE then the string will be trimmed (left and/or right)

`split_string` either TRUE or FALSE. If TRUE then the character string will be split using the *split_separator* as delimiter. The user can also specify multiple delimiters.

`split_separator` a character string specifying the character delimiter(s)

`remove_stopwords` either TRUE, FALSE or a character vector of user defined stop words. If TRUE then by using the *language* parameter the corresponding stop words vector will be uploaded.

`language` a character string which defaults to english. If the *remove_stopwords* parameter is TRUE then the corresponding stop words vector will be uploaded. Available languages are *afrikaans*, *arabic*, *armenian*, *basque*, *bengali*, *breton*, *bulgarian*, *catalan*, *croatian*, *czech*, *danish*, *dutch*, *english*, *estonian*, *finnish*, *french*, *galician*, *german*, *greek*, *hausa*, *hebrew*, *hindi*, *hungarian*, *indonesian*, *irish*, *italian*, *latvian*, *marathi*, *norwegian*, *persian*, *polish*, *portuguese*, *romanian*, *russian*, *slovak*, *slovenian*, *somalia*, *spanish*, *swahili*, *swedish*, *turkish*, *yoruba*, *zulu*

`min_num_char` an integer specifying the minimum number of characters to keep. If the *min_num_char* is greater than 1 then character strings with more than 1 characters will be returned

`max_num_char` an integer specifying the maximum number of characters to keep. The *max_num_char* should be less than or equal to *Inf* (in this function the Inf value translates to a word-length of 1000000000)

`stemmer` a character string specifying the stemming method. One of the following *porter2_stemmer*, *ngram_sequential*, *ngram_overlap*. See details for more information.

`min_n_gram` an integer specifying the minimum number of n-grams. The minimum number of min_n_gram is 1.

`max_n_gram` an integer specifying the maximum number of n-grams. The minimum number of max_n_gram is 1.

`skip_n_gram` an integer specifying the number of skip-n-grams. The minimum number of skip_n_gram is 1. The skip_n_gram gives the (max.) n-grams using the *skip_distance* parameter. If *skip_n_gram* is greater than 1 then both *min_n_gram* and *max_n_gram* should be set to 1.

`skip_distance` an integer specifying the skip distance between the words. The minimum value for the skip distance is 0, in which case simple n-grams will be returned.

`n_gram_delimiter` a character string specifying the n-gram delimiter (applies to both n-gram and skip-n-gram cases)

`concat_delimiter` either NULL or a character string specifying the delimiter to use in order to concatenate the end-vector of character strings to a single character string (recommended in case that the end-vector should be saved to a file)

`path_2folder` a character string specifying the path to the folder where the file(s) will be saved

`stemmer_ngram` a numeric value greater than 1. Applies to both *ngram_sequential* and *ngram_overlap* methods. In case of *ngram_sequential* the first stemmer_ngram characters will be picked, whereas in the case of *ngram_overlap* the overlapping stemmer_ngram characters will be build.

`stemmer_gamma` a float number greater or equal to 0.0. Applies only to *ngram_sequential*. Is a threshold value, which defines how much frequency deviation of two N-grams is acceptable. It is kept either zero or to a minimum value.

`stemmer_truncate`
a numeric value greater than 0. Applies only to *ngram_sequential*. The ngram_sequential is modified to use relative frequencies (float numbers between 0.0 and 1.0 for the ngrams of a specific word in the corpus) and the stemmer_truncate parameter controls the number of rounding digits for the ngrams of the word. The main purpose was to give the same relative frequency to words appearing approximately the same on the corpus.

`stemmer_batches`
a numeric value greater than 0. Applies only to *ngram_sequential*. Splits the corpus into batches with the option to run the batches in multiple threads.

`threads` an integer specifying the number of cores to run in parallel

`save_2single_file`
either TRUE or FALSE. If TRUE then the output data will be saved in a single file. Otherwise the data will be saved in multiple files with incremented enumeration

`increment_batch_nr`
a numeric value. The enumeration of the output files will start from the *increment_batch_nr*. If the *save_2single_file* parameter is TRUE then the *increment_batch_nr* parameter won't be taken into consideration.

`vocabulary_path_file`
either NULL or a character string specifying the output file where the vocabulary should be saved (after tokenization and transformation is applied). Applies to the *vocabulary_accumulator* method.

`vocabulary_path_folder`
either NULL or a character string specifying the output folder where the vocabulary batches should be saved (after tokenization and transformation is applied). Applies to the *big_text_tokenizer* method.

`max_num_chars` a numeric value to limit the words of the output vocabulary to a maximum number of characters (applies to the *vocabulary_accumulator* function)

Format

An object of class `R6ClassGenerator` of length 24.

Details

the *big_text_splitter* function splits a text file into sub-text-files using either the batches parameter (big-text-splitter-bytes) or both the batches and the end_query parameter (big-text-splitter-query). The end_query parameter (if not NULL) should be a character string specifying a word that appears repeatedly at the end of each line in the text file.

the *big_text_parser* function parses text files from an input folder and saves those processed files to an output folder. The *big_text_parser* is appropriate for files with a structure using the start- and end- query parameters.

the *big_text_tokenizer* function tokenizes and transforms the text files of a folder and saves those files to either a folder or a single file. There is also the option to save a frequency vocabulary of those transformed tokens to a file.

the *vocabulary_accumulator* function takes the resulted vocabulary files of the *big_text_tokenizer* and returns the vocabulary sums sorted in decreasing order. The parameter *max_num_chars* limits the number of the corpus using the number of characters of each word.

The *ngram_sequential* or *ngram_overlap* stemming method applies to each single batch and not to the whole corpus of the text file. Thus, it is possible that the stems of the same words for randomly selected batches might differ.

Methods

```
big_tokenize_transform$new(verbose = FALSE)
--------------
big_text_splitter(input_path_file = NULL, output_path_folder = NULL, end_query = NULL, batches = NU

--------------
big_text_parser(input_path_folder = NULL, output_path_folder = NULL, start_query = NULL, end_query

--------------
big_text_tokenizer(input_path_folder = NULL, batches = NULL, read_file_delimiter = " ", to_lower =

--------------
vocabulary_accumulator(input_path_folder = NULL, vocabulary_path_file = NULL, max_num_chars = 100)
```

Examples

```
library(textTinyR)


# fs <- big_tokenize_transform$new(verbose = FALSE)

#---------------
# file splitter:
#---------------

# fs$big_text_splitter(input_path_file = "input.txt",

#                      output_path_folder = "/folder/output/",

#                      end_query = "endword", batches = 5,

#                      trimmed_line = FALSE)


#-------------
```

```
# file parser:
#-------------

# fs$big_text_parser(input_path_folder = "/folder/output/",

#                    output_path_folder = "/folder/parser/",

#                    start_query = "startword", end_query = "endword",

#                    min_lines = 1, trimmed_line = TRUE)


#----------------
# file tokenizer:
#----------------


# fs$big_text_tokenizer(input_path_folder = "/folder/parser/",

#                       batches = 5, split_string=TRUE,

#                       to_lower = TRUE, trim_token = TRUE,

#                       max_num_char = 100, remove_stopwords = TRUE,

#                       stemmer = "porter2_stemmer", threads = 1,

#                       path_2folder="/folder/output_token/",

#                       vocabulary_path_folder="/folder/VOCAB/")

#-------------------
# vocabulary counts:
#-------------------


# fs$vocabulary_accumulator(input_path_folder = "/folder/VOCAB/",

#                           vocabulary_path_file = "/folder/vocab.txt",

#                           max_num_chars = 50)
```

`bytes_converter` *bytes converter of a text file (KB, MB or GB)*

Description

bytes converter of a text file (KB, MB or GB)

Usage

```
bytes_converter(input_path_file = NULL, unit = "MB")
```

Arguments

input_path_file	a character string specifying the path to the input file
unit	a character string specifying the unit. One of *KB*, *MB*, *GB*

Value

a number

Examples

```
library(textTinyR)

# bc = bytes_converter(input_path_file = 'some_file.txt', unit = "MB")
```

---

cosine_distance *cosine distance of two character strings (each string consists of more than one words)*

Description

cosine distance of two character strings (each string consists of more than one words)

Usage

```
cosine_distance(sentence1, sentence2, split_separator = " ")
```

Arguments

sentence1	a character string consisting of multiple words
sentence2	a character string consisting of multiple words
split_separator	a character string specifying the delimiter(s) to split the sentence

Value

a float number

Examples

```
library(textTinyR)

sentence1 = 'this is one sentence'

sentence2 = 'this is a similar sentence'

cds = cosine_distance(sentence1, sentence2)
```

---

dense_2sparse *convert a dense matrix to a sparse matrix*

Description

convert a dense matrix to a sparse matrix

Usage

```
dense_2sparse(dense_mat)
```

Arguments

dense_mat	a dense matrix

Value

a sparse matrix

Examples

```
library(textTinyR)

tmp = matrix(sample(0:1, 100, replace = TRUE), 10, 10)

sp_mat = dense_2sparse(tmp)
```

dice_distance *dice similarity of words using n-grams*

Description

dice similarity of words using n-grams

Usage

```
dice_distance(word1, word2, n_grams = 2)
```

Arguments

word1	a character string
word2	a character string
n_grams	a value specifying the consecutive n-grams of the words

Value

a float number

Examples

```
library(textTinyR)

word1 = 'one_word'

word2 = 'two_words'

dts = dice_distance(word1, word2, n_grams = 2)
```

levenshtein_distance *levenshtein distance of two words*

Description

levenshtein distance of two words

Usage

```
levenshtein_distance(word1, word2)
```

Arguments

word1	a character string
word2	a character string

Value

a float number

Examples

```
library(textTinyR)

word1 = 'one_word'

word2 = 'two_words'

lvs = levenshtein_distance(word1, word2)
```

`load_sparse_binary` *load a sparse matrix in binary format*

Description

load a sparse matrix in binary format

Usage

```
load_sparse_binary(file_name = "save_sparse.mat")
```

Arguments

`file_name`	a character string specifying the binary file

Value

loads a sparse matrix from a file

Examples

```
library(textTinyR)

# load_sparse_binary(file_name = "save_sparse.mat")
```

matrix_sparsity *sparsity percentage of a sparse matrix*

Description

sparsity percentage of a sparse matrix

Usage

```
matrix_sparsity(sparse_matrix)
```

Arguments

sparse_matrix	a sparse matrix

Value

a numeric value (percentage)

Examples

```
library(textTinyR)

tmp = matrix(sample(0:1, 100, replace = TRUE), 10, 10)

sp_mat = dense_2sparse(tmp)

dbl = matrix_sparsity(sp_mat)
```

read_characters *read a specific number of characters from a text file*

Description

read a specific number of characters from a text file

Usage

```
read_characters(input_file = NULL, characters = 100, write_2file = "")
```

Arguments

input_file	a character string specifying a valid path to a text file
characters	a numeric value specifying the number of characters to read
write_2file	either an empty string ("") or a character string specifying a valid output file to write the subset of the input file

Examples

```
library(textTinyR)

# txfl = read_characters(input_file = 'input.txt', characters = 100)
```

`read_rows` *read a specific number of rows from a text file*

Description

read a specific number of rows from a text file

Usage

```
read_rows(input_file = NULL, read_delimiter = "\n", rows = 100,
  write_2file = "")
```

Arguments

`input_file`	a character string specifying a valid path to a text file
`read_delimiter`	a character string specifying the row delimiter of the text file
`rows`	a numeric value specifying the number of rows to read
`write_2file`	either "" or a character string specifying a valid output file to write the subset of the input file

Examples

```
library(textTinyR)

# txfl = read_rows(input_file = 'input.txt', rows = 100)
```

`save_sparse_binary` *save a sparse matrix in binary format*

Description

save a sparse matrix in binary format

Usage

```
save_sparse_binary(sparse_matrix, file_name = "save_sparse.mat")
```

Arguments

sparse_matrix	a sparse matrix
file_name	a character string specifying the binary file

Value

writes the sparse matrix to a file

Examples

```
library(textTinyR)

tmp = matrix(sample(0:1, 100, replace = TRUE), 10, 10)

sp_mat = dense_2sparse(tmp)

# save_sparse_binary(sp_mat, file_name = "save_sparse.mat")
```

---

sparse_Means *RowMens and colMeans for a sparse matrix*

Description

RowMens and colMeans for a sparse matrix

Usage

```
sparse_Means(sparse_matrix, rowMeans = FALSE)
```

Arguments

sparse_matrix	a sparse matrix
rowMeans	either TRUE or FALSE. If TRUE then the row-means will be calculated, otherwise the column-means

Value

a vector with either the row- or the column-sums of the matrix

Examples

```
library(textTinyR)

tmp = matrix(sample(0:1, 100, replace = TRUE), 10, 10)

sp_mat = dense_2sparse(tmp)

spsm = sparse_Means(sp_mat, rowMeans = FALSE)
```

sparse_Sums *RowSums and colSums for a sparse matrix*

Description

RowSums and colSums for a sparse matrix

Usage

```
sparse_Sums(sparse_matrix, rowSums = FALSE)
```

Arguments

sparse_matrix	a sparse matrix
rowSums	either TRUE or FALSE. If TRUE then the row-sums will be calculated, otherwise the column-sums

Value

a vector with either the row- or the column-sums of the matrix

Examples

```
library(textTinyR)

tmp = matrix(sample(0:1, 100, replace = TRUE), 10, 10)

sp_mat = dense_2sparse(tmp)

spsm = sparse_Sums(sp_mat, rowSums = FALSE)
```

sparse_term_matrix *Term matrices and statistics (document-term-matrix, term-document-matrix)*

Description

Term matrices and statistics (document-term-matrix, term-document-matrix)

Usage

```
# utl <- sparse_term_matrix$new(vector_data = NULL, file_data = NULL,

#                                       document_term_matrix = TRUE)
```

Arguments

vector_data	either NULL or a character vector of documents
file_data	either NULL or a valid character path to a text file
document_term_matrix	either TRUE or FALSE. If TRUE then a document-term-matrix will be returned, otherwise a term-document-matrix
sort_terms	either TRUE or FALSE specifying if the initial terms should be sorted (so that the output sparse matrix is sorted in alphabetical order)
to_lower	either TRUE or FALSE. If TRUE the character string will be converted to lower case
remove_char	a string specifying the specific characters that should be removed from a text file. If the *remove_char* is "" then no removal of characters take place
to_upper	either TRUE or FALSE. If TRUE the character string will be converted to upper case
utf_locale	the language specific locale to use in case that either the *to_lower* or the *to_upper* parameter is TRUE and the text file language is other than english. For instance if the language of a text file is greek then the *utf_locale* parameter should be *'el_GR.UTF-8'* (*language_country.encoding*). A wrong utf-locale does not raise an error, however the runtime of the function increases.
remove_punctuation_string	either TRUE or FALSE. If TRUE then the punctuation of the character string will be removed (applies before the split function)
remove_punctuation_vector	either TRUE or FALSE. If TRUE then the punctuation of the vector of the character strings will be removed (after the string split has taken place)
remove_numbers	either TRUE or FALSE. If TRUE then any numbers in the character string will be removed
trim_token	either TRUE or FALSE. If TRUE then the string will be trimmed (left and/or right)
split_string	either TRUE or FALSE. If TRUE then the character string will be split using the *split_separator* as delimiter. The user can also specify multiple delimiters.
split_separator	a character string specifying the character delimiter(s)
remove_stopwords	either TRUE, FALSE or a character vector of user defined stop words. If TRUE then by using the *language* parameter the corresponding stop words vector will be uploaded.
language	a character string which defaults to english. If the *remove_stopwords* parameter is TRUE then the corresponding stop words vector will be uploaded. Available languages are *afrikaans*, *arabic*, *armenian*, *basque*, *bengali*, *breton*, *bulgarian*, *catalan*, *croatian*, *czech*, *danish*, *dutch*, *english*, *estonian*, *finnish*, *french*, *galician*, *german*, *greek*, *hausa*, *hebrew*, *hindi*, *hungarian*, *indonesian*, *irish*, *italian*, *latvian*, *marathi*, *norwegian*, *persian*, *polish*, *portuguese*, *romanian*, *russian*, *slovak*, *slovenian*, *somalia*, *spanish*, *swahili*, *swedish*, *turkish*, *yoruba*, *zulu*

`min_num_char` an integer specifying the minimum number of characters to keep. If the *min_num_char* is greater than 1 then character strings with more than 1 characters will be returned

`max_num_char` an integer specifying the maximum number of characters to keep. The *max_num_char* should be less than or equal to *Inf* (in this function the Inf value translates to a word-length of 1000000000)

`stemmer` a character string specifying the stemming method. Available method is the *porter2_stemmer*. See details for more information.

`min_n_gram` an integer specifying the minimum number of n-grams. The minimum number of min_n_gram is 1.

`max_n_gram` an integer specifying the maximum number of n-grams. The minimum number of max_n_gram is 1.

`skip_n_gram` an integer specifying the number of skip-n-grams. The minimum number of skip_n_gram is 1. The skip_n_gram gives the (max.) n-grams using the *skip_distance* parameter. If *skip_n_gram* is greater than 1 then both *min_n_gram* and *max_n_gram* should be set to 1.

`skip_distance` an integer specifying the skip distance between the words. The minimum value for the skip distance is 0, in which case simple n-grams will be returned.

`n_gram_delimiter` a character string specifying the n-gram delimiter (applies to both n-gram and skip-n-gram cases)

`print_every_rows` a numeric value greater than 1 specifying the print intervals. Frequent output in the R session can slow down the function in case of big files.

`normalize` either NULL or one of 'l1' or 'l2' normalization.

`tf_idf` either TRUE or FALSE. If TRUE then the term-frequency-inverse-document-frequency will be returned

`threads` an integer specifying the number of cores to run in parallel

`verbose` either TRUE or FALSE. If TRUE then information will be printed out

`sparsity_thresh` a float number between 0.0 and 1.0 specifying the sparsity threshold in the *Term_Matrix_Adjust* function

`Terms` a character vector specifying the character strings for which the associations will be calculated (*term_associations* function)

`keep_terms` either NULL or a numeric value specifying the number of terms to keep (both in *term_associations* and *most_frequent_terms* functions)

Format

An object of class `R6ClassGenerator` of length 24.

Details

the *Term_Matrix* function takes either a character vector of strings or a text file and after tokenization and transformation returns either a document-term-matrix or a term-document-matrix

the *triplet_data* function returns the triplet data, which is used internally (in c++), to construct the Term Matrix. The triplet data could be usefull for secondary purposes, such as in word vector representations.

the *Term_Matrix_Adjust* function removes sparse terms from a sparse matrix using a sparsity threshold

the *term_associations* function finds the associations between the given terms (Terms argument) and all the other terms in the corpus by calculating their correlation. There is also the option to keep a specific number of terms from the output table using the *keep_terms* parameter.

the *most_frequent_terms* function returns the most frequent terms of the corpus using the output of the sparse matrix. The user has the option to keep a specific number of terms from the output table using the *keep_terms* parameter.

Stemming of the english language is done using the porter2-stemmer, for details see https://github.com/smassung/porter2_stemmer

Methods

```
sparse_term_matrix$new(vector_data = NULL, file_data = NULL, document_term_matrix = TRUE)

--------------

Term_Matrix(sort_terms = FALSE, to_lower = FALSE, to_upper = FALSE, utf_locale = "", remove_char =

--------------

triplet_data()
--------------

Term_Matrix_Adjust(sparsity_thresh = 1.0)
--------------

term_associations(Terms = NULL, keep_terms = NULL, threads = 1, verbose = FALSE)

--------------

most_frequent_terms(keep_terms = NULL, threads = 1, verbose = FALSE)
```

Examples

```
library(textTinyR)


# sm <- sparse_term_matrix$new(file_data = "/folder/my_data.txt",

#                              document_term_matrix = TRUE)
```

```
#--------------
# term matrix :
#--------------

# sm$Term_Matrix(sort_terms = TRUE, to_lower = TRUE,

#                trim_token = TRUE, split_string = TRUE,

#                remove_stopwords = TRUE, normalize = 'l1',

#                stemmer = 'porter2_stemmer', threads = 1 )

#---------------
# triplet data :
#---------------

# sm$triplet_data()

#-------------------------
# removal of sparse terms:
#-------------------------

# sm$Term_Matrix_Adjust(sparsity_thresh = 0.995)


#-----------------------------------------------
# associations between terms of a sparse matrix:
#-----------------------------------------------


# sm$term_associations(Terms = c("word", "sentence"), keep_terms = 10)


#-------------------------------------------
# most frequent terms using the sparse matrix:
#-------------------------------------------


# sm$most_frequent_terms(keep_terms = 10, threads = 1)
```

`text_file_parser` *text file parser*

Description

text file parser

Usage

```
text_file_parser(input_path_file = NULL, output_path_file = NULL,
```

```
  start_query = NULL, end_query = NULL, min_lines = 1,
  trimmed_line = FALSE, verbose = FALSE)
```

Arguments

input_path_file	a character string specifying the path to the input file
output_path_file	a character string specifying the path to the output file
start_query	a character string. The *start_query* is the first word of the subset of the data and should appear frequently at the beginning of each line in the text file.
end_query	a character string. The *end_query* is the last word of the subset of the data and should appear frequently at the end of each line in the text file.
min_lines	a numeric value specifying the minimum number of lines. For instance if min_lines = 2, then only subsets of text with more than 1 lines will be kept.
trimmed_line	either TRUE or FALSE. If FALSE then each line of the text file will be trimmed both sides before applying the start_query and end_query
verbose	either TRUE or FALSE. If TRUE then information will be printed in the console

Details

The text file should have a structure (such as an xml-structure), so that subsets can be extracted using the *start_query* and *end_query* parameters.

Examples

```
library(textTinyR)

# fp = text_file_parser(input_path_file = '/folder/input_data.txt',

#                       output_path_file = '/folder/output_data.txt',

#                       start_query = 'word_a', end_query = 'word_w',

#                       min_lines = 1, trimmed_line = FALSE)
```

---

tokenize_transform_text

String tokenization and transformation (character string or path to a file)

Description

String tokenization and transformation (character string or path to a file)

Usage

```
tokenize_transform_text(object = NULL, batches = NULL,
  read_file_delimiter = "\n", to_lower = FALSE, to_upper = FALSE,
  utf_locale = "", remove_char = "", remove_punctuation_string = FALSE,
  remove_punctuation_vector = FALSE, remove_numbers = FALSE,
  trim_token = FALSE, split_string = FALSE,
  split_separator = " \r\n\t.,;:()?!//", remove_stopwords = FALSE,
  language = "english", min_num_char = 1, max_num_char = Inf,
  stemmer = NULL, min_n_gram = 1, max_n_gram = 1, skip_n_gram = 1,
  skip_distance = 0, n_gram_delimiter = " ", concat_delimiter = NULL,
  path_2folder = "", stemmer_ngram = 4, stemmer_gamma = 0,
  stemmer_truncate = 3, stemmer_batches = 1, threads = 1,
  vocabulary_path_file = NULL, verbose = FALSE)
```

Arguments

`object`	either a character string (text data) or a character-string-path to a file (for big .txt files it's recommended to use a path to a file).
`batches`	a numeric value. If the *batches* parameter is not NULL then the *object* parameter should be a valid path to a file and the *path_2folder* parameter should be a valid path to a folder. The batches parameter should be used in case of small to medium data sets (for zero memory consumption). For big data sets the *big_tokenize_transform* R6 class and especially the *big_text_tokenizer* function should be used.
`read_file_delimiter`	the delimiter to use when the input file will be red (for instance a tab-delimiter or a new-line delimiter).
`to_lower`	either TRUE or FALSE. If TRUE the character string will be converted to lower case
`to_upper`	either TRUE or FALSE. If TRUE the character string will be converted to upper case
`utf_locale`	the language specific locale to use in case that either the *to_lower* or the *to_upper* parameter is TRUE and the text file language is other than english. For instance if the language of a text file is greek then the *utf_locale* parameter should be *'el_GR.UTF-8'* (*language_country.encoding*). A wrong utf-locale does not raise an error, however the runtime of the function increases.
`remove_char`	a character string with specific characters that should be removed from the text file. If the *remove_char* is "" then no removal of characters take place
`remove_punctuation_string`	either TRUE or FALSE. If TRUE then the punctuation of the character string will be removed (applies before the split function)
`remove_punctuation_vector`	either TRUE or FALSE. If TRUE then the punctuation of the vector of the character strings will be removed (after the string split has taken place)
`remove_numbers`	either TRUE or FALSE. If TRUE then any numbers in the character string will be removed

`trim_token` either TRUE or FALSE. If TRUE then the string will be trimmed (left and/or right)

`split_string` either TRUE or FALSE. If TRUE then the character string will be split using the *split_separator* as delimiter. The user can also specify multiple delimiters.

`split_separator` a character string specifying the character delimiter(s)

`remove_stopwords` either TRUE, FALSE or a character vector of user defined stop words. If TRUE then by using the *language* parameter the corresponding stop words vector will be uploaded.

`language` a character string which defaults to english. If the *remove_stopwords* parameter is TRUE then the corresponding stop words vector will be uploaded. Available languages are *afrikaans*, *arabic*, *armenian*, *basque*, *bengali*, *breton*, *bulgarian*, *catalan*, *croatian*, *czech*, *danish*, *dutch*, *english*, *estonian*, *finnish*, *french*, *galician*, *german*, *greek*, *hausa*, *hebrew*, *hindi*, *hungarian*, *indonesian*, *irish*, *italian*, *latvian*, *marathi*, *norwegian*, *persian*, *polish*, *portuguese*, *romanian*, *russian*, *slovak*, *slovenian*, *somalia*, *spanish*, *swahili*, *swedish*, *turkish*, *yoruba*, *zulu*

`min_num_char` an integer specifying the minimum number of characters to keep. If the *min_num_char* is greater than 1 then character strings with more than 1 characters will be returned

`max_num_char` an integer specifying the maximum number of characters to keep. The *max_num_char* should be less than or equal to *Inf* (in this function the Inf value translates to a word-length of 1000000000)

`stemmer` a character string specifying the stemming method. One of the following *porter2_stemmer*, *ngram_sequential*, *ngram_overlap*. See details for more information.

`min_n_gram` an integer specifying the minimum number of n-grams. The minimum number of min_n_gram is 1.

`max_n_gram` an integer specifying the maximum number of n-grams. The minimum number of max_n_gram is 1.

`skip_n_gram` an integer specifying the number of skip-n-grams. The minimum number of skip_n_gram is 1. The skip_n_gram gives the (max.) n-grams using the *skip_distance* parameter. If *skip_n_gram* is greater than 1 then both *min_n_gram* and *max_n_gram* should be set to 1.

`skip_distance` an integer specifying the skip distance between the words. The minimum value for the skip distance is 0, in which case simple n-grams will be returned.

`n_gram_delimiter` a character string specifying the n-gram delimiter (applies to both n-gram and skip-n-gram cases)

`concat_delimiter` either NULL or a character string specifying the delimiter to use in order to concatenate the end-vector of character strings to a single character string (recommended in case that the end-vector should be saved to a file)

`path_2folder` a character string specifying the path to the folder where the file(s) will be saved

`stemmer_ngram` a numeric value greater than 1. Applies to both *ngram_sequential* and *ngram_overlap* methods. In case of *ngram_sequential* the first *n* characters will be picked, whereas in the case of *ngram_overlap* the overlapping stemmer_ngram characters will be build.

`stemmer_gamma` a float number greater or equal to 0.0. Applies only to *ngram_sequential*. Is a threshold value, which defines how much frequency deviation of two N-grams is acceptable. It is kept either zero or to a minimum value.

`stemmer_truncate`
a numeric value greater than 0. Applies only to *ngram_sequential*. The ngram_sequential is modified to use relative frequencies (float numbers between 0.0 and 1.0 for the ngrams of a specific word in the corpus) and the stemmer_truncate parameter controls the number of rounding digits for the ngrams of the word. The main purpose was to give the same relative frequency to words appearing approximately the same on the corpus.

`stemmer_batches`
a numeric value greater than 0. Applies only to *ngram_sequential*. Splits the corpus into batches with the option to run the batches in multiple threads.

`threads` an integer specifying the number of cores to run in parallel

`vocabulary_path_file`
either NULL or a character string specifying the output path to a file where the vocabulary should be saved once the text is tokenized

`verbose` either TRUE or FALSE. If TRUE then information will be printed out

Details

It is memory efficient to read the data using a *path file* in case of a big file, rather than importing the data in the R-session and then calling the *tokenize_transform_text* function.

It is memory efficient to give a *path_2folder* in case that a big file should be saved, rather than return the vector of all character strings in the R-session.

The *skip-grams* are a generalization of n-grams in which the components (typically words) need not to be consecutive in the text under consideration, but may leave gaps that are skipped over. They provide one way of overcoming the *data sparsity problem* found with conventional n-gram analysis.

Many character string pre-processing functions (such as the *utf-locale* or the *split-string* function) are based on the *boost* library (http://www.boost.org/).

Stemming of the english language is done using the porter2-stemmer, for details see https://github.com/smassung/porter2_stemmer

N-gram stemming is language independent and supported by the following two functions:

ngram_overlap The *ngram_overlap* stemming method is based on *N-Gram Morphemes for Retrieval, Paul McNamee and James Mayfield*, http://clef.isti.cnr.it/2007/working_notes/mcnameeCLEF2007.pdf

ngram_sequential The *ngram_sequential* stemming method is a modified version based on *Generation, Implementation and Appraisal of an N-gram based Stemming Algorithm, B. P. Pande, Pawan Tamta, H. S. Dhami*, https://arxiv.org/pdf/1312.4824.pdf

The list of stop-words in the available languages was downloaded from the following link, https://github.com/6/stopwords-json

Value

a character vector

Examples

```
library(textTinyR)

token_str = "CONVERT to lower, remove.. punctuation11234, trim token and split "

res = tokenize_transform_text(object = token_str, to_lower = TRUE, split_string = TRUE)
```

---

tokenize_transform_vec_docs

String tokenization and transformation (vector of documents)

Description

String tokenization and transformation (vector of documents)

Usage

```
tokenize_transform_vec_docs(object = NULL, as_token = FALSE,
  to_lower = FALSE, to_upper = FALSE, utf_locale = "", remove_char = "",
  remove_punctuation_string = FALSE, remove_punctuation_vector = FALSE,
  remove_numbers = FALSE, trim_token = FALSE, split_string = FALSE,
  split_separator = " \r\n\t.,;:()?!//", remove_stopwords = FALSE,
  language = "english", min_num_char = 1, max_num_char = Inf,
  stemmer = NULL, min_n_gram = 1, max_n_gram = 1, skip_n_gram = 1,
  skip_distance = 0, n_gram_delimiter = " ", concat_delimiter = NULL,
  path_2folder = "", threads = 1, vocabulary_path_file = NULL,
  verbose = FALSE)
```

Arguments

object	a character string vector of documents
as_token	if TRUE then the output of the function is a list of (split) token. Otherwise is a vector of character strings (sentences)
to_lower	either TRUE or FALSE. If TRUE the character string will be converted to lower case
to_upper	either TRUE or FALSE. If TRUE the character string will be converted to upper case
utf_locale	the language specific locale to use in case that either the *to_lower* or the *to_upper* parameter is TRUE and the text file language is other than english. For instance if the language of a text file is greek then the *utf_locale* parameter should be *'el_GR.UTF-8'* (*language_country.encoding*). A wrong utf-locale does not raise an error, however the runtime of the function increases.

`remove_char` a character string with specific characters that should be removed from the text file. If the *remove_char* is "" then no removal of characters take place

`remove_punctuation_string`
either TRUE or FALSE. If TRUE then the punctuation of the character string will be removed (applies before the split function)

`remove_punctuation_vector`
either TRUE or FALSE. If TRUE then the punctuation of the vector of the character strings will be removed (after the string split has taken place)

`remove_numbers` either TRUE or FALSE. If TRUE then any numbers in the character string will be removed

`trim_token` either TRUE or FALSE. If TRUE then the string will be trimmed (left and/or right)

`split_string` either TRUE or FALSE. If TRUE then the character string will be split using the *split_separator* as delimiter. The user can also specify multiple delimiters.

`split_separator`
a character string specifying the character delimiter(s)

`remove_stopwords`
either TRUE, FALSE or a character vector of user defined stop words. If TRUE then by using the *language* parameter the corresponding stop words vector will be uploaded.

`language` a character string which defaults to english. If the *remove_stopwords* parameter is TRUE then the corresponding stop words vector will be uploaded. Available languages are *afrikaans*, *arabic*, *armenian*, *basque*, *bengali*, *breton*, *bulgarian*, *catalan*, *croatian*, *czech*, *danish*, *dutch*, *english*, *estonian*, *finnish*, *french*, *galician*, *german*, *greek*, *hausa*, *hebrew*, *hindi*, *hungarian*, *indonesian*, *irish*, *italian*, *latvian*, *marathi*, *norwegian*, *persian*, *polish*, *portuguese*, *romanian*, *russian*, *slovak*, *slovenian*, *somalia*, *spanish*, *swahili*, *swedish*, *turkish*, *yoruba*, *zulu*

`min_num_char` an integer specifying the minimum number of characters to keep. If the *min_num_char* is greater than 1 then character strings with more than 1 characters will be returned

`max_num_char` an integer specifying the maximum number of characters to keep. The *max_num_char* should be less than or equal to *Inf* (in this function the Inf value translates to a word-length of 1000000000)

`stemmer` a character string specifying the stemming method. Available method is the *porter2_stemmer*. See details for more information.

`min_n_gram` an integer specifying the minimum number of n-grams. The minimum number of min_n_gram is 1.

`max_n_gram` an integer specifying the maximum number of n-grams. The minimum number of max_n_gram is 1.

`skip_n_gram` an integer specifying the number of skip-n-grams. The minimum number of skip_n_gram is 1. The skip_n_gram gives the (max.) n-grams using the *skip_distance* parameter. If *skip_n_gram* is greater than 1 then both *min_n_gram* and *max_n_gram* should be set to 1.

`skip_distance` an integer specifying the skip distance between the words. The minimum value for the skip distance is 0, in which case simple n-grams will be returned.

`n_gram_delimiter`
: a character string specifying the n-gram delimiter (applies to both n-gram and skip-n-gram cases)

`concat_delimiter`
: either NULL or a character string specifying the delimiter to use in order to concatenate the end-vector of character strings to a single character string (recommended in case that the end-vector should be saved to a file)

`path_2folder`
: a character string specifying the path to the folder where the file(s) will be saved

`threads`
: an integer specifying the number of cores to run in parallel

`vocabulary_path_file`
: either NULL or a character string specifying the output path to a file where the vocabulary should be saved once the text is tokenized

`verbose`
: either TRUE or FALSE. If TRUE then information will be printed out

Details

It is memory efficient to give a *path_2folder* in case that a big file should be saved, rather than return the vector of all character strings in the R-session.

The *skip-grams* are a generalization of n-grams in which the components (typically words) need not to be consecutive in the text under consideration, but may leave gaps that are skipped over. They provide one way of overcoming the *data sparsity problem* found with conventional n-gram analysis.

Many character string pre-processing functions (such as the *utf-locale* or the *split-string* function) are based on the *boost* library (http://www.boost.org/).

Stemming of the english language is done using the porter2-stemmer, for details see https://github.com/smassung/porter2_stemmer

The list of stop-words in the available languages was downloaded from the following link, https://github.com/6/stopwords-json

Value

a character vector

Examples

```
library(textTinyR)

token_doc_vec = c("CONVERT to lower", "remove.. punctuation11234", "trim token and split ")

res = tokenize_transform_vec_docs(object = token_doc_vec, to_lower = TRUE, split_string = TRUE)
```

token_stats *token statistics*

Description

token statistics

Usage

```
# utl <- token_stats$new(x_vec = NULL, path_2folder = NULL, path_2file = NULL,

#                                file_delimiter = ' ', n_gram_delimiter = "_")
```

Arguments

x_vec	either NULL or a string character vector
path_2folder	either NULL or a valid path to a folder (each file in the folder should include words separated by a delimiter)
path_2file	either NULL or a valid path to a file
file_delimiter	either NULL or a character string specifying the file delimiter
n_gram_delimiter	either NULL or a character string specifying the n-gram delimiter. It is used in the *collocation_words* function
subset	either NULL or a vector specifying the subset of data to keep (number of rows of the *print_frequency* function)
number	a numeric value for the *print_count_character* function. All words with number of characters equal to the *number* parameter will be returned.
word	a character string for the *print_collocations* and *print_prob_next* functions
dice_n_gram	a numeric value specifying the n-gram for the dice method of the *string_dissimilarity_matrix* function
method	a character string specifying the method to use in the *string_dissimilarity_matrix* function. One of *dice*, *levenshtein* or *cosine*.
split_separator	a character string specifying the string split separator if method equal *cosine* in the *string_dissimilarity_matrix* function. The *cosine* method uses sentences, so for a sentence : "this_is_a_word_sentence" the *split_separator* should be "_"
dice_thresh	a float number to use to threshold the data if method is *dice* in the *string_dissimilarity_matrix* function. It takes values between 0.0 and 1.0. The closer the thresh is to 0.0 the more values of the dissimilarity matrix will take the value of 1.0.
upper	either TRUE or FALSE. If TRUE then both lower and upper parts of the dissimilarity matrix of the *string_dissimilarity_matrix* function will be shown. Otherwise the upper part will be filled with NA's

`diagonal`	either TRUE or FALSE. If TRUE then the diagonal of the dissimilarity matrix of the *string_dissimilarity_matrix* function will be shown. Otherwise the diagonal will be filled with NA's
`threads`	a numeric value specifying the number of cores to use in parallel in the *string_dissimilarity_matrix* function
`n_grams`	a numeric value specifying the n-grams in the *look_up_table* function
`n_gram`	a character string specifying the n-gram to use in the *print_words_lookup_tbl* function

Format

An object of class `R6ClassGenerator` of length 24.

Details

the *path_2vector* function returns the words of a *folder* or *file* to a vector (using the *file_delimiter* to input the data). Usage: read a vocabulary from a text file

the *freq_distribution* function returns a named-unsorted vector frequency_distribution in R for EITHER a *folder*, a *file* OR a character string *vector*. A specific subset of the result can be retrieved using the *print_frequency* function

the *count_character* function returns the number of characters for each word of the corpus for EITHER a *folder*, a *file* OR a character string *vector*. A specific number of character words can be retrieved using the *print_count_character* function

the *collocation_words* function returns a co-occurence frequency table for n-grams for EITHER a *folder*, a *file* OR a character string *vector*. A collocation is defined as a sequence of two or more consecutive words, that has characteristics of a syntactic and semantic unit, and whose exact and unambiguous meaning or connotation cannot be derived directly from the meaning or connotation of its components (http://nlp.stanford.edu/fsnlp/promo/colloc.pdf, page 172). The input to the function should be text n-grams separated by a delimiter (for instance 3- or 4-ngrams). I can retrieve a specific frequency table by using the *print_collocations* function

the *string_dissimilarity_matrix* function returns a string-dissimilarity-matrix using either the *dice*, *levenshtein* or *cosine* distance. The input can be a character string *vector* only. In case that the method is *dice* then the dice-coefficient (similarity) is calculated between two strings for a specific number of character n-grams (*dice_n_gram*).

the *look_up_table* returns a look-up-list where the list-names are the n-grams and the list-vectors are the words associated with those n-grams. The words for each n-gram can be retrieved using the *print_words_lookup_tbl* function. The input can be a character string *vector* only.

Methods

```
token_stats$new(x_vec = NULL, path_2folder = NULL, path_2file = NULL, file_delimiter = ' ', n_gram_
```

```
--------------
```

```
path_2vector()
```

```
--------------
```

```
freq_distribution()
--------------
print_frequency(subset = NULL)
--------------
count_character()
--------------
print_count_character(number = NULL)
--------------
collocation_words()
--------------
print_collocations(word = NULL)
--------------
string_dissimilarity_matrix(dice_n_gram = 2, method = "dice", split_separator = " ", dice_thresh =

--------------
look_up_table(n_grams = NULL)
--------------
print_words_lookup_tbl(n_gram = NULL)
```

Examples

```
library(textTinyR)

expl = c('one_word_token', 'two_words_token', 'three_words_token', 'four_words_token')

tk <- token_stats$new(x_vec = expl, path_2folder = NULL, path_2file = NULL)

#-------------------------
# frequency distribution:
#-------------------------

tk$freq_distribution()

# tk$print_frequency()


#------------------
# count characters:
#------------------

cnt <- tk$count_character()

# tk$print_count_character(number = 4)
```

```
#----------------------
# collocation of words:
#----------------------

col <- tk$collocation_words()

# tk$print_collocations(word = 'five')


#-----------------------------
# string dissimilarity matrix:
#-----------------------------

dism <- tk$string_dissimilarity_matrix(method = 'levenshtein')


#---------------------
# build a look-up-table:
#---------------------

lut <- tk$look_up_table(n_grams = 3)

# tk$print_words_lookup_tbl(n_gram = 'e_w')
```

`utf_locale` *utf-locale for the available languages*

Description

utf-locale for the available languages

Usage

```
utf_locale(language = "english")
```

Arguments

`language`	a character string specifying the language for which the utf-locale should be returned

Details

This is a limited list of language-locale. The locale depends mostly on the text input.

Value

a utf locale

Examples

```
library(textTinyR)

utf_locale(language = "english")
```

---

vocabulary_parser *returns the vocabulary counts for small or medium (xml and not only) files*

Description

returns the vocabulary counts for small or medium (xml and not only) files

Usage

```
vocabulary_parser(input_path_file = NULL, start_query = NULL,
  end_query = NULL, vocabulary_path_file = NULL, min_lines = 1,
  trimmed_line = FALSE, to_lower = FALSE, to_upper = FALSE,
  utf_locale = "", max_num_char = Inf, remove_char = "",
  remove_punctuation_string = FALSE, remove_punctuation_vector = FALSE,
  remove_numbers = FALSE, trim_token = FALSE, split_string = FALSE,
  split_separator = " \r\n\t.,;:()?!//", remove_stopwords = FALSE,
  language = "english", min_num_char = 1, stemmer = NULL,
  min_n_gram = 1, max_n_gram = 1, skip_n_gram = 1, skip_distance = 0,
  n_gram_delimiter = " ", threads = 1, verbose = FALSE)
```

Arguments

`input_path_file`
: a character string specifying a valid path to the input file

`start_query`
: a character string. The *start_query* is the first word of the subset of the data and should appear frequently at the beginning of each line in the text file.

`end_query`
: a character string. The *end_query* is the last word of the subset of the data and should appear frequently at the end of each line in the text file.

`vocabulary_path_file`
: a character string specifying the output file where the vocabulary should be saved (after tokenization and transformation is applied).

`min_lines`
: a numeric value specifying the minimum number of lines. For instance if min_lines = 2, then only subsets of text with more than 1 lines will be kept.

`trimmed_line`
: either TRUE or FALSE. If FALSE then each line of the text file will be trimmed both sides before applying the start_query and end_query

`to_lower`
: either TRUE or FALSE. If TRUE the character string will be converted to lower case

`to_upper` either TRUE or FALSE. If TRUE the character string will be converted to upper case

`utf_locale` the language specific locale to use in case that either the *to_lower* or the *to_upper* parameter is TRUE and the text file language is other than english. For instance if the language of a text file is greek then the *utf_locale* parameter should be *'el_GR.UTF-8'* (*language_country.encoding*). A wrong utf-locale does not raise an error, however the runtime of the function increases.

`max_num_char` an integer specifying the maximum number of characters to keep. The *max_num_char* should be less than or equal to *Inf* (in this function the Inf value translates to a word-length of 1000000000)

`remove_char` a character string with specific characters that should be removed from the text file. If the *remove_char* is "" then no removal of characters take place

`remove_punctuation_string`
either TRUE or FALSE. If TRUE then the punctuation of the character string will be removed (applies before the split function)

`remove_punctuation_vector`
either TRUE or FALSE. If TRUE then the punctuation of the vector of the character strings will be removed (after the string split has taken place)

`remove_numbers` either TRUE or FALSE. If TRUE then any numbers in the character string will be removed

`trim_token` either TRUE or FALSE. If TRUE then the string will be trimmed (left and/or right)

`split_string` either TRUE or FALSE. If TRUE then the character string will be split using the *split_separator* as delimiter. The user can also specify multiple delimiters.

`split_separator`
a character string specifying the character delimiter(s)

`remove_stopwords`
either TRUE, FALSE or a character vector of user defined stop words. If TRUE then by using the *language* parameter the corresponding stop words vector will be uploaded.

`language` a character string which defaults to english. If the *remove_stopwords* parameter is TRUE then the corresponding stop words vector will be uploaded. Available languages are *afrikaans*, *arabic*, *armenian*, *basque*, *bengali*, *breton*, *bulgarian*, *catalan*, *croatian*, *czech*, *danish*, *dutch*, *english*, *estonian*, *finnish*, *french*, *galician*, *german*, *greek*, *hausa*, *hebrew*, *hindi*, *hungarian*, *indonesian*, *irish*, *italian*, *latvian*, *marathi*, *norwegian*, *persian*, *polish*, *portuguese*, *romanian*, *russian*, *slovak*, *slovenian*, *somalia*, *spanish*, *swahili*, *swedish*, *turkish*, *yoruba*, *zulu*

`min_num_char` an integer specifying the minimum number of characters to keep. If the *min_num_char* is greater than 1 then character strings with more than 1 characters will be returned

`stemmer` a character string specifying the stemming method. Available method is the *porter2_stemmer*. See details for more information.

`min_n_gram` an integer specifying the minimum number of n-grams. The minimum number of min_n_gram is 1.

max_n_gram	an integer specifying the maximum number of n-grams. The minimum number of max_n_gram is 1.
skip_n_gram	an integer specifying the number of skip-n-grams. The minimum number of skip_n_gram is 1. The skip_n_gram gives the (max.) n-grams using the *skip_distance* parameter. If *skip_n_gram* is greater than 1 then both *min_n_gram* and *max_n_gram* should be set to 1.
skip_distance	an integer specifying the skip distance between the words. The minimum value for the skip distance is 0, in which case simple n-grams will be returned.
n_gram_delimiter	a character string specifying the n-gram delimiter (applies to both n-gram and skip-n-gram cases)
threads	an integer specifying the number of cores to run in parallel
verbose	either TRUE or FALSE. If TRUE then information will be printed in the console

Details

The text file should have a structure (such as an xml-structure), so that subsets can be extracted using the *start_query* and *end_query* parameters

For big files the *vocabulary_accumulator* method of the *big_tokenize_transform* class is appropriate

Stemming of the english language is done using the porter2-stemmer, for details see https://github.com/smassung/porter2_stemmer

Examples

```
library(textTinyR)

# vps = vocabulary_parser(input_path_file = '/folder/input_data.txt',

#                         start_query = 'start_word', end_query = 'end_word',

#                         vocabulary_path_file = '/folder/vocab.txt',

#                         to_lower = TRUE, split_string = TRUE)
```

Index