

# Package ‘textTinyR’

July 25, 2018

**Type** Package

**Title** Text Processing for Small or Big Data Files

**Version** 1.1.2

**Date** 2018-07-25

**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** It offers functions for splitting, parsing, tokenizing and creating a vocabulary for big text data files. Moreover, it includes functions for building a document-term matrix and extracting information from those (term-associations, most frequent terms). It also embodies functions for calculating token statistics (collocations, look-up tables, string dissimilarities) and functions to work with sparse matrices. Lastly, it includes functions for Word Vector Representations (i.e. 'GloVe', 'fasttext') and incorporates functions for the calculation of (pair-wise) text document dissimilarities. 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 a C++11 compiler

**LazyData** TRUE

**Depends** R(>= 3.2.3), Matrix

**Imports** Rcpp (>= 0.12.10), R6, data.table, utils

**LinkingTo** Rcpp, RcppArmadillo (>= 0.7.8), BH

**Suggests** testthat, covr, knitr, rmarkdown

**VignetteBuilder** knitr

**RoxygenNote** 6.0.1

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2018-07-25 07:30:02 UTC

**R topics documented:**

big_tokenize_transform . . . . .	2
bytes_converter . . . . .	7
cluster_frequency . . . . .	8
cosine_distance . . . . .	9
COS_TEXT . . . . .	10
Count_Rows . . . . .	11
dense_2sparse . . . . .	11
dice_distance . . . . .	12
dims_of_word_vecs . . . . .	13
Doc2Vec . . . . .	13
JACCARD_DICE . . . . .	16
levenshtein_distance . . . . .	17
load_sparse_binary . . . . .	17
matrix_sparsity . . . . .	18
read_characters . . . . .	19
read_rows . . . . .	19
save_sparse_binary . . . . .	20
select_predictors . . . . .	21
sparse_Means . . . . .	22
sparse_Sums . . . . .	23
sparse_term_matrix . . . . .	23
TEXT_DOC DISSIM . . . . .	28
text_file_parser . . . . .	29
text_intersect . . . . .	31
tokenize_transform_text . . . . .	32
tokenize_transform_vec_docs . . . . .	36
token_stats . . . . .	38
utf_locale . . . . .	42
vocabulary_parser . . . . .	43
<b>Index</b>	<b>46</b>

---

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 <i>start_query</i> 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 <i>end_query</i> 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 <i>min_lines</i> = 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 <i>start_query</i> and <i>end_query</i>
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 ( <i>big_text_splitter</i> function) or will be used internally for low memory processing ( <i>big_text_tokenizer</i> function).
read_file_delimiter	the delimiter to use when the input file will be read (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 <i>remove_char</i> 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 <i>to_lower</i> or the <i>to_upper</i> 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 <i>utf_locale</i> parameter should be 'el_GR.UTF-8' ( <i>language_country.encoding</i> ). 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 <i>split_separator</i> 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 <i>language</i> parameter the corresponding stop words vector will be uploaded.
language	a character string which defaults to english. If the <i>remove_stopwords</i> parameter is TRUE then the corresponding stop words vector will be uploaded. Available languages are <i>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</i>
min_num_char	an integer specifying the minimum number of characters to keep. If the <i>min_num_char</i> 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 <i>max_num_char</i> should be less than or equal to <i>Inf</i> (in this function the <i>Inf</i> value translates to a word-length of 1000000000)
stemmer	a character string specifying the stemming method. One of the following <i>porter2_stemmer, ngram_sequential, ngram_overlap</i> . See details for more information.
min_n_gram	an integer specifying the minimum number of n-grams. The minimum number of <i>min_n_gram</i> is 1.
max_n_gram	an integer specifying the maximum number of n-grams. The minimum number of <i>max_n_gram</i> is 1.
skip_n_gram	an integer specifying the number of skip-n-grams. The minimum number of <i>skip_n_gram</i> is 1. The <i>skip_n_gram</i> gives the (max.) n-grams using the <i>skip_distance</i> parameter. If <i>skip_n_gram</i> is greater than 1 then both <i>min_n_gram</i> and <i>max_n_gram</i> 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 <i>ngram_sequential</i> and <i>ngram_overlap</i> methods. In case of <i>ngram_sequential</i> the first <i>stemmer_ngram</i> characters will be picked, whereas in the case of <i>ngram_overlap</i> the overlapping <i>stemmer_ngram</i> characters will be build.

stemmer_gamma	a float number greater or equal to 0.0. Applies only to <i>ngram_sequential</i> . 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 <i>ngram_sequential</i> . The <i>ngram_sequential</i> 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 <i>stemmer_truncate</i> 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 <i>ngram_sequential</i> . 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 <i>increment_batch_nr</i> . If the <i>save_2single_file</i> parameter is TRUE then the <i>increment_batch_nr</i> 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 <i>vocabulary_accumulator</i> 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 <i>big_text_tokenizer</i> method.
max_num_chars	a numeric value to limit the words of the output vocabulary to a maximum number of characters (applies to the <i>vocabulary_accumulator</i> 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")
```

---

cluster_frequency	<i>Frequencies of an existing cluster object</i>
-------------------	--

---

**Description**

Frequencies of an existing cluster object

**Usage**

```
cluster_frequency(tokenized_list_text, cluster_vector, verbose = FALSE)
```

**Arguments**

```
tokenized_list_text  a list of tokenized text documents. This can be the result of the textTinyR::tokenize_transform_vec_docs
                    function with the as_token parameter set to TRUE (the token object of the out-
                    put)
cluster_vector       a numeric vector. This can be the result of the ClusterR::KMeans_rcpp function
                    (the clusters object of the output)
verbose              either TRUE or FALSE. If TRUE then information will be printed out in the R
                    session.
```

**Details**

This function takes a list of tokenized text and a numeric vector of clusters and returns the sorted frequency of each cluster. The length of the *tokenized\_list\_text* object must be equal to the length of the *cluster\_vector* object



**Value**

a list of data.tables

**Examples**

```
library(textTinyR)

tok_lst = list(c('the', 'the', 'tokens', 'of', 'first', 'document'),
              c('the', 'tokens', 'of', 'of', 'second', 'document'),
              c('the', 'tokens', 'of', 'third', 'third', 'document'))

vec_clust = rep(1:6, 3)

res = cluster_frequency(tok_lst, vec_clust)
```

---

cosine_distance	<i>cosine distance of two character strings (each string consists of more than one words)</i>
-----------------	---

---

**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'

cfs = cosine_distance(sentence1, sentence2)
```

---

`COS_TEXT`*Cosine similarity for text documents*

---

**Description**

Cosine similarity for text documents

**Usage**

```
COS_TEXT(text_vector1 = NULL, text_vector2 = NULL, threads = 1,  
         separator = " ")
```

**Arguments**

<code>text_vector1</code>	a character string vector representing text documents (it should have the same length as the <code>text_vector2</code> )
<code>text_vector2</code>	a character string vector representing text documents (it should have the same length as the <code>text_vector1</code> )
<code>threads</code>	a numeric value specifying the number of cores to run in parallel
<code>separator</code>	specifies the separator used between words of each character string in the text vectors

**Details**

The function calculates the *cosine* distance between pairs of text sequences of two character string vectors

**Value**

a numeric vector

**Examples**

```
library(textTinyR)  
  
vec1 = c('use this', 'function to compute the')  
  
vec2 = c('cosine distance', 'between text sequences')  
  
out = COS_TEXT(text_vector1 = vec1, text_vector2 = vec2, separator = " ")
```

---

Count_Rows	<i>Number of rows of a file</i>
------------	---------------------------------

---

**Description**

Number of rows of a file

**Usage**

```
Count_Rows(PATH, verbose = FALSE)
```

**Arguments**

PATH	a character string specifying the path to a file
verbose	either TRUE or FALSE

**Details**

This function returns the number of rows for a file. It doesn't load the data in memory.

**Value**

a numeric value

**Examples**

```
library(textTinyR)

PATH = system.file("example_files", "word_vecs.txt", package = "textTinyR")

num_rows = Count_Rows(PATH)
```

---

dense_2sparse	<i>convert a dense matrix to a sparse matrix</i>
---------------	--

---

**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	<i>dice similarity of words using n-grams</i>
---------------	---

---

**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)
```

---

dims_of_word_vecs	<i>dimensions of a word vectors file</i>
-------------------	--

---

**Description**

dimensions of a word vectors file

**Usage**

```
dims_of_word_vecs(input_file = NULL, read_delimiter = "\n")
```

**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

**Details**

This function takes a valid path to a file and a file delimiter as input and estimates the dimensions of the word vectors by using the first row of the file.

**Value**

a numeric value

**Examples**

```
library(textTinyR)

PATH = system.file("example_files", "word_vecs.txt", package = "textTinyR")

dimensions = dims_of_word_vecs(input_file = PATH)
```

---

Doc2Vec	<i>Conversion of text documents to word-vector-representation features ( Doc2Vec )</i>
---------	--

---

**Description**

Conversion of text documents to word-vector-representation features ( Doc2Vec )

**Usage**

```
# utl <- Doc2Vec$new(token_list = NULL, word_vector_FILE = NULL,
#                   print_every_rows = 10000, verbose = FALSE,
#                   copy_data = FALSE)
```

**Arguments**

**token\_list** either NULL or a list of tokenized text documents

**word\_vector\_FILE** a valid path to a text file, where the word-vectors are saved

**print\_every\_rows** a numeric value greater than 1 specifying the print intervals. Frequent output in the R session can slow down the function especially in case of big files.

**verbose** either TRUE or FALSE. If TRUE then information will be printed out in the R session.

**method** a character string specifying the method to use. One of *sum\_sqrt*, *min\_max\_norm* or *idf*. See the details section for more information.

**global\_term\_weights** either NULL or the output of the *global\_term\_weights* method of the *textTinyR* package. See the details section for more information.

**threads** a numeric value specifying the number of cores to run in parallel

**copy\_data** either TRUE or FALSE. If FALSE then a pointer will be created and no copy of the initial data takes place (memory efficient especially for big datasets). This is an alternative way to pre-process the data.

**Format**

An object of class *R6ClassGenerator* of length 24.

**Details**

the *pre\_processed\_wv* method should be used after the initialization of the *Doc2Vec* class, if the *copy\_data* parameter is set to TRUE, in order to inspect the pre-processed word-vectors.

The *global\_term\_weights* method is part of the *sparse\_term\_matrix* R6 class of the *textTinyR* package. One can come to the correct *global\_term\_weights* by using the *sparse\_term\_matrix* class and by setting the *tf\_idf* parameter to FALSE and the *normalize* parameter to NULL. In *Doc2Vec* class, if method equals to *idf* then the *global\_term\_weights* parameter should not be equal to NULL.

Explanation of the various *methods* :

**sum\_sqrt** Assuming that a single sublist of the token list will be taken into consideration : the wordvectors of each word of the sublist of tokens will be accumulated to a vector equal to the length of the wordvector (INITIAL\_WORD\_VECTOR). Then a scalar will be computed using this INITIAL\_WORD\_VECTOR in the following way : the INITIAL\_WORD\_VECTOR will be raised to the power of 2.0, then the resulted wordvector will be summed and the square-root will be calculated. The INITIAL\_WORD\_VECTOR will be divided by the resulted scalar

**min\_max\_norm** Assuming that a single sublist of the token list will be taken into consideration : the wordvectors of each word of the sublist of tokens will be first *min-max* normalized and then will be accumulated to a vector equal to the length of the initial wordvector

**idf** Assuming that a single sublist of the token list will be taken into consideration : the word-vector of each term in the sublist will be multiplied with the corresponding *idf* of the *global weights term*

There might be slight differences in the output data for each method depending on the input value of the *copy\_data* parameter (if it's either TRUE or FALSE).

## Value

a matrix

## Methods

```
Doc2Vec$new(token_list = NULL, word_vector_FILE = NULL, print_every_rows = 10000, verbose = FALSE,
```

```
-----
```

```
doc2vec_methods(method = "sum_sqrt", global_term_weights = NULL, threads = 1)
```

```
-----
```

```
pre_processed_wv()
```

## Examples

```
library(textTinyR)

#-----
# tokenized text in form of a list
#-----

tok_text = list(c('the', 'result', 'of'), c('doc2vec', 'are', 'vector', 'features'))

#-----
# path to the word vectors
#-----

PATH = system.file("example_files", "word_vecs.txt", package = "textTinyR")

init = Doc2Vec$new(token_list = tok_text, word_vector_FILE = PATH)

out = init$doc2vec_methods(method = "sum_sqrt")
```

---

**JACCARD\_DICE***Jaccard or Dice similarity for text documents*

---

**Description**

Jaccard or Dice similarity for text documents

**Usage**

```
JACCARD_DICE(token_list1 = NULL, token_list2 = NULL, method = "jaccard",
             threads = 1)
```

**Arguments**

<code>token_list1</code>	a list of tokenized text documents (it should have the same length as the <code>token_list2</code> )
<code>token_list2</code>	a list of tokenized text documents (it should have the same length as the <code>token_list1</code> )
<code>method</code>	a character string specifying the similarity metric. One of 'jaccard', 'dice'
<code>threads</code>	a numeric value specifying the number of cores to run in parallel

**Details**

The function calculates either the *jaccard* or the *dice* distance between pairs of tokenized text of two lists

**Value**

a numeric vector

**Examples**

```
library(textTinyR)

lst1 = list(c('use', 'this', 'function', 'to'), c('either', 'compute', 'the', 'jaccard'))
lst2 = list(c('or', 'the', 'dice', 'distance'), c('for', 'two', 'same', 'sized', 'lists'))
out = JACCARD_DICE(token_list1 = lst1, token_list2 = lst2, method = 'jaccard', threads = 1)
```



---

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	<i>sparsity percentage of a sparse matrix</i>
-----------------	---

---

**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	<i>read a specific number of characters from a text file</i>
-----------------	--

---

**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	<i>read a specific number of rows from a text file</i>
-----------	--

---

**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")
```

---

select_predictors	<i>Exclude highly correlated predictors</i>
-------------------	---

---

### Description

Exclude highly correlated predictors

### Usage

```
select_predictors(response_vector, predictors_matrix,  
  response_lower_thresh = 0.1, predictors_upper_thresh = 0.75,  
  threads = 1, verbose = FALSE)
```

### Arguments

response_vector	a numeric vector (the length should be equal to the rows of the <i>predictors_matrix</i> parameter)
predictors_matrix	a numeric matrix (the rows should be equal to the length of the <i>response_vector</i> parameter)
response_lower_thresh	a numeric value. This parameter allows the user to keep all the predictors having a correlation with the response <i>greater</i> than the <i>response_lower_thresh</i> value.
predictors_upper_thresh	a numeric value. This parameter allows the user to keep all the predictors having a correlation comparing to the other predictors <i>less</i> than the <i>predictors_upper_thresh</i> value.
threads	a numeric value specifying the number of cores to run in parallel
verbose	either TRUE or FALSE. If TRUE then information will be printed out in the R session.

### Details

The function works in the following way : The correlation of the predictors with the response is first calculated and the resulted correlations are sorted in decreasing order. Then iteratively predictors with correlation higher than the *predictors\_upper\_thresh* value are removed by favoring those predictors which are more correlated with the response variable. If the *response\_lower\_thresh* value is greater than 0.0 then only predictors having a correlation higher than or equal to the *response\_lower\_thresh* value will be kept, otherwise they will be excluded. This function returns the indices of the *predictors* and is useful in case of multicollinearity.

### Value

a vector of column-indices

**Examples**

```
library(textTinyR)

set.seed(1)
resp = runif(100)

set.seed(2)
col = runif(100)

matr = matrix(c(col, col^4, col^6, col^8, col^10), nrow = 100, ncol = 5)

out = select_predictors(resp, matr, predictors_upper_thresh = 0.75)
```

---

sparse_Means	<i>RowMeans and colMeans for a sparse matrix</i>
--------------	--

---

**Description**

RowMeans 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)
```



**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 <i>remove_char</i> 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 <i>to_lower</i> or the <i>to_upper</i> 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 <i>utf_locale</i> parameter should be ' <i>el_GR.UTF-8</i> ' ( <i>language_country.encoding</i> ). 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 <i>split_separator</i> 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 <i>language</i> parameter the corresponding stop words vector will be uploaded.
language	a character string which defaults to english. If the <i>remove_stopwords</i> parameter is TRUE then the corresponding stop words vector will be uploaded. Available languages are <i>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</i>



min_num_char	an integer specifying the minimum number of characters to keep. If the <i>min_num_char</i> 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 <i>max_num_char</i> should be less than or equal to <i>Inf</i> (in this function the <i>Inf</i> value translates to a word-length of 1000000000)
stemmer	a character string specifying the stemming method. Available method is the <i>porter2_stemmer</i> . See details for more information.
min_n_gram	an integer specifying the minimum number of n-grams. The minimum number of <i>min_n_gram</i> is 1.
max_n_gram	an integer specifying the maximum number of n-grams. The minimum number of <i>max_n_gram</i> is 1.
skip_n_gram	an integer specifying the number of skip-n-grams. The minimum number of <i>skip_n_gram</i> is 1. The <i>skip_n_gram</i> gives the (max.) n-grams using the <i>skip_distance</i> parameter. If <i>skip_n_gram</i> is greater than 1 then both <i>min_n_gram</i> and <i>max_n_gram</i> 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 <i>Term_Matrix_Adjust</i> function
Terms	a character vector specifying the character strings for which the associations will be calculated ( <i>term_associations</i> function )
keep_terms	either NULL or a numeric value specifying the number of terms to keep ( both in <i>term_associations</i> and <i>most_frequent_terms</i> 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 *global\_term\_weights* function returns a list of length two. The first sublist includes the *terms* and the second sublist the *global-term-weights*. The *tf\_idf* parameter should be set to FALSE and the *normalize* parameter to NULL. This function is normally used in conjunction with word-vector-embeddings.

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](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()

-----

global_term_weights()

-----

Term_Matrix_Adjust(sparsity_thresh = 1.0)

-----

term_associations(Terms = NULL, keep_terms = NULL, 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()

#-----
# global-term-weights :
#-----

# sm$global_term_weights()

#-----
# 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\_DOC\_DISSIM      *Dissimilarity calculation of text documents*

---

### Description

Dissimilarity calculation of text documents

### Usage

```
TEXT_DOC_DISSIM(first_matr = NULL, second_matr = NULL,
  method = "euclidean", batches = NULL, threads = 1, verbose = FALSE)
```

### Arguments

<code>first_matr</code>	a numeric matrix where each row represents a text document ( has same dimensions as the <i>second_matr</i> )
<code>second_matr</code>	a numeric matrix where each row represents a text document ( has same dimensions as the <i>first_matr</i> )
<code>method</code>	a dissimilarity metric in form of a character string. One of <i>euclidean</i> , <i>manhattan</i> , <i>chebyshev</i> , <i>canberra</i> , <i>braycurtis</i> , <i>pearson_correlation</i> , <i>cosine</i> , <i>simple_matching_coefficient</i> , <i>hamming</i> , <i>jaccard_coefficient</i> , <i>Rao_coefficient</i>
<code>batches</code>	a numeric value specifying the number of batches
<code>threads</code>	a numeric value specifying the number of cores to run in parallel
<code>verbose</code>	either TRUE or FALSE. If TRUE then information will be printed in the console

### Details

Row-wise dissimilarity calculation of text documents. The text document sequences should be converted to numeric matrices using for instance LSI (Latent Semantic Indexing). If the numeric matrices are too big to be pre-processed, then one should use the *batches* parameter to split the data in batches before applying one of the dissimilarity metrics. For parallelization (*threads*) OpenMP will be used.

### Value

a numeric vector

**Examples**

```
## Not run:

library(textTinyR)

# example input LSI matrices (see details section)
#-----

set.seed(1)
LSI_matrix1 = matrix(runif(10000), 100, 100)

set.seed(2)
LSI_matrix2 = matrix(runif(10000), 100, 100)

txt_out = TEXT_DOC_DISSIM(first_matr = LSI_matrix1,
                           second_matr = LSI_matrix2, 'euclidean')

## End(Not run)
```

---

text_file_parser	<i>text file parser</i>
------------------	-------------------------

---

**Description**

text file parser

**Usage**

```
text_file_parser(input_path_file = NULL, output_path_file = "",
                 start_query = NULL, end_query = NULL, min_lines = 1,
                 trimmed_line = FALSE, verbose = FALSE)
```

**Arguments**

**input\_path\_file** either a path to an input file or a vector of character strings ( normally the latter would represent ordered lines of a text file in form of a character vector )

**output\_path\_file** either an empty character string ("" ) or a character string specifying a path to an output file ( it applies only if the *input\_path\_file* parameter is a valid path to a file )

**start\_query** a character string or a vector of character strings. The *start\_query* (if it's a single character string) 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 or a vector of character strings. The <i>end_query</i> (if it's a single character string) 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 ( applies only if the <i>input_path_file</i> is a valid path to a file) . For instance if min_lines = 2, then only subsets of text with more than 1 lines will be pre-processed.
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 ( the same applies in case of a vector of character strings)

### Examples

```
library(textTinyR)

# In case that the 'input_path_file' is a valid path
#-----

# 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)

# In case that the 'input_path_file' is a character vector of strings
#-----

# PATH_url = "https://FILE.xml"
# con = url(PATH_url, method = "libcurl")
# tmp_dat = read.delim(con, quote = "\"", comment.char = "", stringsAsFactors = FALSE)
# vec_docs = unlist(lapply(1:length(as.vector(tmp_dat[, 1])), function(x)
#
#                       trimws(tmp_dat[x, 1], which = "both")))
# parse_data = text_file_parser(input_path_file = vec_docs,
#
#                               start_query = c("<query1>", "<query2>", "<query3>"),
#
#                               end_query = c("</query1>", "</query2>", "</query3>"),
```

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

---

```
text_intersect           intersection of words or letters in tokenized text
```

---

### Description

intersection of words or letters in tokenized text

### Usage

```
# utl <- text_intersect$new(token_list1 = NULL, token_list2 = NULL)
```

### Arguments

token_list1	a list, where each sublist is a tokenized text sequence ( <i>token_list1</i> should be of same length with <i>token_list2</i> )
token_list2	a list, where each sublist is a tokenized text sequence ( <i>token_list2</i> should be of same length with <i>token_list1</i> )
distinct	either TRUE or FALSE. If TRUE then the intersection of <i>distinct</i> words (or letters) will be taken into account
letters	either TRUE or FALSE. If TRUE then the intersection of letters in the text sequences will be computed

### Format

An object of class R6ClassGenerator of length 24.

### Details

This class includes methods for text or character intersection. If both *distinct* and *letters* are FALSE then the simple (count or ratio) word intersection will be computed.

### Value

a numeric vector

### Methods

```
text_intersect$new(file_data = NULL)
-----
count_intersect(distinct = FALSE, letters = FALSE)
-----
ratio_intersect(distinct = FALSE, letters = FALSE)
```

**References**

<https://www.kaggle.com/c/home-depot-product-search-relevance/discussion/20427> by Igor Buinyi

**Examples**

```
library(textTinyR)

tok1 = list(c('compare', 'this', 'text'),
            c('and', 'this', 'text'))

tok2 = list(c('with', 'another', 'set'),
            c('of', 'text', 'documents'))

init = text_intersect$new(tok1, tok2)

init$count_intersect(distinct = TRUE, letters = FALSE)

init$ratio_intersect(distinct = FALSE, letters = TRUE)
```

---

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 <i>batches</i> parameter is not NULL then the <i>object</i> parameter should be a valid path to a file and the <i>path_2folder</i> parameter should be a valid path to a folder. The <i>batches</i> parameter should be used in case of small to medium data sets (for zero memory consumption). For big data sets the <i>big_tokenize_transform</i> R6 class and especially the <i>big_text_tokenizer</i> function should be used.
read_file_delimiter	the delimiter to use when the input file will be read (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 <i>to_lower</i> or the <i>to_upper</i> 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 <i>utf_locale</i> parameter should be 'el_GR.UTF-8' ( <i>language_country.encoding</i> ). 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 <i>remove_char</i> 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 <i>split_separator</i> 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 <i>language</i> parameter the corresponding stop words vector will be uploaded.
language	a character string which defaults to english. If the <i>remove_stopwords</i> parameter is TRUE then the corresponding stop words vector will be uploaded. Available languages are <i>afrikaans</i> , <i>arabic</i> , <i>armenian</i> , <i>basque</i> , <i>bengali</i> , <i>breton</i> , <i>bulgarian</i> ,

*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*

<code>min_num_char</code>	an integer specifying the minimum number of characters to keep. If the <code>min_num_char</code> is greater than 1 then character strings with more than 1 characters will be returned
<code>max_num_char</code>	an integer specifying the maximum number of characters to keep. The <code>max_num_char</code> should be less than or equal to <i>Inf</i> (in this function the <i>Inf</i> value translates to a word-length of 1000000000)
<code>stemmer</code>	a character string specifying the stemming method. One of the following <i>porter2_stemmer</i> , <i>ngram_sequential</i> , <i>ngram_overlap</i> . See details for more information.
<code>min_n_gram</code>	an integer specifying the minimum number of n-grams. The minimum number of <code>min_n_gram</code> is 1.
<code>max_n_gram</code>	an integer specifying the maximum number of n-grams. The minimum number of <code>max_n_gram</code> is 1.
<code>skip_n_gram</code>	an integer specifying the number of skip-n-grams. The minimum number of <code>skip_n_gram</code> is 1. The <code>skip_n_gram</code> gives the (max.) n-grams using the <i>skip_distance</i> parameter. If <code>skip_n_gram</code> is greater than 1 then both <code>min_n_gram</code> and <code>max_n_gram</code> should be set to 1.
<code>skip_distance</code>	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.
<code>n_gram_delimiter</code>	a character string specifying the n-gram delimiter (applies to both n-gram and skip-n-gram cases)
<code>concat_delimiter</code>	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)
<code>path_2folder</code>	a character string specifying the path to the folder where the file(s) will be saved
<code>stemmer_ngram</code>	a numeric value greater than 1. Applies to both <i>ngram_sequential</i> and <i>ngram_overlap</i> methods. In case of <i>ngram_sequential</i> the first <i>n</i> characters will be picked, whereas in the case of <i>ngram_overlap</i> the overlapping stemmer_ngram characters will be build.
<code>stemmer_gamma</code>	a float number greater or equal to 0.0. Applies only to <i>ngram_sequential</i> . 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.
<code>stemmer_truncate</code>	a numeric value greater than 0. Applies only to <i>ngram_sequential</i> . The <i>ngram_sequential</i> 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 <code>stemmer_truncate</code> 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 <i>ngram_sequential</i> . 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](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](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. Dhani, <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 <i>to_lower</i> or the <i>to_upper</i> 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 <i>utf_locale</i> parameter should be <i>'el_GR.UTF-8'</i> ( <i>language_country.encoding</i> ). 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 <i>remove_char</i> 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 <i>split_separator</i> 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 <i>language</i> parameter the corresponding stop words vector will be uploaded.
language	a character string which defaults to english. If the <i>remove_stopwords</i> parameter is TRUE then the corresponding stop words vector will be uploaded. Available languages are <i>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</i>
min_num_char	an integer specifying the minimum number of characters to keep. If the <i>min_num_char</i> 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 <i>max_num_char</i> should be less than or equal to <i>Inf</i> (in this function the <i>Inf</i> value translates to a word-length of 1000000000)
stemmer	a character string specifying the stemming method. Available method is the <i>porter2_stemmer</i> . See details for more information.
min_n_gram	an integer specifying the minimum number of n-grams. The minimum number of <i>min_n_gram</i> is 1.
max_n_gram	an integer specifying the maximum number of n-grams. The minimum number of <i>max_n_gram</i> is 1.
skip_n_gram	an integer specifying the number of skip-n-grams. The minimum number of <i>skip_n_gram</i> is 1. The <i>skip_n_gram</i> gives the (max.) n-grams using the <i>skip_distance</i> parameter. If <i>skip_n_gram</i> is greater than 1 then both <i>min_n_gram</i> and <i>max_n_gram</i> 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](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	<i>token statistics</i>
-------------	-------------------------

---

### 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 <i>collocation_words</i> function
subset	either NULL or a vector specifying the subset of data to keep (number of rows of the <i>print_frequency</i> function)
number	a numeric value for the <i>print_count_character</i> function. All words with number of characters equal to the <i>number</i> parameter will be returned.
word	a character string for the <i>print_collocations</i> and <i>print_prob_next</i> functions
dice_n_gram	a numeric value specifying the n-gram for the dice method of the <i>string_dissimilarity_matrix</i> function
method	a character string specifying the method to use in the <i>string_dissimilarity_matrix</i> function. One of <i>dice</i> , <i>levenshtein</i> or <i>cosine</i> .
split_separator	a character string specifying the string split separator if method equal <i>cosine</i> in the <i>string_dissimilarity_matrix</i> function. The <i>cosine</i> method uses sentences, so for a sentence : "this_is_a_word_sentence" the <i>split_separator</i> should be "_"
dice_thresh	a float number to use to threshold the data if method is <i>dice</i> in the <i>string_dissimilarity_matrix</i> 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 <i>string_dissimilarity_matrix</i> 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 <i>string_dissimilarity_matrix</i> 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 <i>string_dissimilarity_matrix</i> function
n_grams	a numeric value specifying the n-grams in the <i>look_up_table</i> function
n_gram	a character string specifying the n-gram to use in the <i>print_words_lookup_tbl</i> 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-occurrence 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-grams ). 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 <i>start_query</i> 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 <i>end_query</i> 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 <i>min_lines</i> = 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 <i>start_query</i> and <i>end_query</i>
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 <i>to_lower</i> or the <i>to_upper</i> 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 <i>utf_locale</i> parameter should be <i>'el_GR.UTF-8'</i> ( <i>language_country.encoding</i> ). 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 <i>max_num_char</i> should be less than or equal to <i>Inf</i> (in this function the <i>Inf</i> 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 <i>remove_char</i> 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 <i>split_separator</i> 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 <i>language</i> parameter the corresponding stop words vector will be uploaded.
language	a character string which defaults to english. If the <i>remove_stopwords</i> parameter is TRUE then the corresponding stop words vector will be uploaded. Available languages are <i>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</i>
min_num_char	an integer specifying the minimum number of characters to keep. If the <i>min_num_char</i> 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 <i>porter2_stemmer</i> . See details for more information.
min_n_gram	an integer specifying the minimum number of n-grams. The minimum number of <i>min_n_gram</i> is 1.
max_n_gram	an integer specifying the maximum number of n-grams. The minimum number of <i>max_n_gram</i> is 1.
skip_n_gram	an integer specifying the number of skip-n-grams. The minimum number of <i>skip_n_gram</i> is 1. The <i>skip_n_gram</i> gives the (max.) n-grams using the <i>skip_distance</i> parameter. If <i>skip_n_gram</i> is greater than 1 then both <i>min_n_gram</i> and <i>max_n_gram</i> 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.



# Index

## \*Topic **datasets**

- big\_tokenize\_transform, 2
- Doc2Vec, 13
- sparse\_term\_matrix, 23
- text\_intersect, 31
- token\_stats, 38

- big\_tokenize\_transform, 2
- bytes\_converter, 7

- cluster\_frequency, 8
- COS\_TEXT, 10
- cosine\_distance, 9
- Count\_Rows, 11

- dense\_2sparse, 11
- dice\_distance, 12
- dims\_of\_word\_vecs, 13
- Doc2Vec, 13

- JACCARD\_DICE, 16

- levenshtein\_distance, 17
- load\_sparse\_binary, 17

- matrix\_sparsity, 18

- read\_characters, 19
- read\_rows, 19

- save\_sparse\_binary, 20
- select\_predictors, 21
- sparse\_Means, 22
- sparse\_Sums, 23
- sparse\_term\_matrix, 23

- TEXT\_DOC DISSIM, 28
- text\_file\_parser, 29
- text\_intersect, 31
- token\_stats, 38
- tokenize\_transform\_text, 32

- tokenize\_transform\_vec\_docs, 36

- utf\_locale, 42

- vocabulary\_parser, 43