Package 'corpustools'

December 13, 2017

Version 0.3.1

Date 2017-12-13

Title Managing, Querying and Analyzing Tokenized Text

Description Provides text analysis in R, focusing on the use of a tokenized text format. In this format, the positions of tokens are maintained, and each token can be annotated (e.g., part-of-speech tags, dependency relations).
Prominent features include advanced Lucene-like querying for specific tokens or contexts (e.g., documents, sentences), similarity statistics for words and documents, exporting to DTM for compatibility with many text analysis packages, and the possibility to reconstruct original text from tokens to facilitate interpretation.

Author Kasper Welbers and Wouter van Atteveldt

Maintainer Kasper Welbers <kasperwelbers@gmail.com>

Depends R (>= 3.2.0), Matrix (>= 1.2), data.table (>= 1.10.4)

Imports memoise, methods, plyr, wordcloud (>= 2.5), SnowballC, stringi, Rcpp (>= 0.12.12), R6

Suggests testthat, quanteda (>= 0.99.9), tm (>= 0.6), topicmodels, RNewsflow, igraph

LinkingTo Rcpp

LazyData true

Encoding UTF-8

SystemRequirements C++11

License GPL-3

URL http://github.com/kasperwelbers/corpustools

RoxygenNote 6.0.1

NeedsCompilation yes

Repository CRAN

Date/Publication 2017-12-13 09:27:42 UTC

R topics documented:

`add_collocation_label` *Choose and add collocation strings based on collocation categories*

Description

Given a collocation category (e.g., named entity ids), this function finds the most frequently occuring string in this category and adds it as a label for the category

Usage

```
add_collocation_label(tc, colloc_id, feature = "token",
  new_feature = sprintf("%s_l", colloc_id), pref_subset = NULL)
```

Arguments

`tc`	a tcorpus object
`colloc_id`	the data column containing the unique id for collocation tokens
`feature`	the name of the feature column
`new_feature`	the name of the new feature column

pref_subset	Optionally, a subset call, to specify a subset that has priority for finding the most frequently occuring string

---

as.tcorpus *Force an object to be a tCorpus class*

Description

Force an object to be a tCorpus class

Usage

```
as.tcorpus(x, ...)
```

Arguments

x	the object to be forced
...	not used

---

as.tcorpus.default *Force an object to be a tCorpus class*

Description

Force an object to be a tCorpus class

Usage

```
## Default S3 method:
as.tcorpus(x, ...)
```

Arguments

x	the object to be forced
...	not used

Examples

```
## Not run:
x = c('First text','Second text')
as.tcorpus(x) ## x is not a tCorpus object

## End(Not run)
```

as.tcorpus.tCorpus *Force an object to be a tCorpus class*

Description

Force an object to be a tCorpus class

Usage

```
## S3 method for class 'tCorpus'
as.tcorpus(x, ...)
```

Arguments

x	the object to be forced
...	not used

Examples

```
tc = create_tcorpus(c('First text', 'Second text'))
as.tcorpus(tc)
```

backbone_filter *Extract the backbone of a network.*

Description

Based on the following paper: Serrano, M. A., Boguna, M., & Vespignani, A. (2009). Extracting the multiscale backbone of complex weighted networks. Proceedings of the National Academy of Sciences, 106(16), 6483-6488.

Usage

```
backbone_filter(g, alpha = 0.05, direction = "none", delete_isolates = T,
  max_vertices = NULL, use_original_alpha = T, k_is_n = F)
```

Arguments

g	A graph in the 'Igraph' format.
alpha	The threshold for the alpha. Can be interpreted similar to a p value (see paper for clarrification).
direction	direction = 'none' can be used for both directed and undirected networks, and is (supposed to be) the disparity filter proposed in Serrano et al. (2009) is used. By setting to 'in' or 'out', the alpha is only calculated for out or in edges. This is an experimental use of the backbone extraction (so beware!) but it seems a logical application.

delete_isolates
: If TRUE, vertices with degree 0 (i.e. no edges) are deleted.

max_vertices
: Optional. Set a maximum number of vertices for the network to be produced. The alpha is then automatically lowered to the point that only the given number of vertices remains connected (degree > 0). This can be usefull if the purpose is to make an interpretation friendly network. See e.g., http://jcom.sissa.it/archive/14/01/JCOM_1401_2015

use_original_alpha
: if max_vertices is not NULL, this determines whether the lower alpha for selecting the top vertices is also used as a threshold for the edges, or whether the original value given in the alpha parameter is used.

k_is_n
: the disparity filter method for backbone extraction uses the number of existing edges (k) for each node, which can be arbitraty if there are many very weak ties, which is often the case in a co-occurence network. By setting k_is_n to TRUE, it is 'assumed' that all nodes are connected, which makes sense from a language model perspective (i.e. probability for co-occurence is never zero)

Value

A graph in the Igraph format

Examples

```
## Not run:
tc = create_tcorpus(sotu_texts, doc_column = 'id')
tc$preprocess('token','feature', remove_stopwords = TRUE, use_stemming = TRUE, min_docfreq = 10)

g = tc$semnet_window('feature', window.size = 10)
igraph::vcount(g)
igraph::ecount(g)
gb = backbone_filter(g, max_vertices = 100)
igraph::vcount(gb)
igraph::ecount(gb)
plot_semnet(gb)

## End(Not run)
```

---

calc_chi2 *Compute the chi^2 statistic for a 2x2 crosstab containing the values [a, b] [c, d]*

Description

Compute the chi^2 statistic for a 2x2 crosstab containing the values [a, b] [c, d]

Usage

```
calc_chi2(a, b, c, d, correct = T, cochrans_criteria = F)
```

Arguments

a	topleft value of the table
b	topright value
c	bottomleft value
d	bottomright value
correct	if TRUE, use yates correction. Can be a vector of length a (i.e. the number of tables)
cochrans_criteria	if TRUE, check if cochrans_criteria indicate that a correction should be used. This overrides the correct parameter

---

corenlp_tokens *coreNLP example sentences*

Description

coreNLP example sentences

Usage

```
data(corenlp_tokens)
```

Format

data.frame

---

create_tcorpus *Create a tCorpus*

Description

Create a tCorpus from raw text input

Usage

```
create_tcorpus(x, ...)

## S3 method for class 'character'
create_tcorpus(x, doc_id = 1:length(x), meta = NULL,
  split_sentences = F, max_sentences = NULL, max_tokens = NULL,
  verbose = F, ...)

## S3 method for class 'factor'
create_tcorpus(x, doc_id = 1:length(x), meta = NULL,
```

```
  split_sentences = F, max_sentences = NULL, max_tokens = NULL,
  verbose = F, ...)

## S3 method for class 'data.frame'
create_tcorpus(x, text_columns = "text",
  doc_column = "doc_id", split_sentences = F, max_sentences = NULL,
  max_tokens = NULL, ...)
```

Arguments

x	main input. can be a character (or factor) vector where each value is a full text, or a data.frame that has a column that contains full texts.
...	not used
doc_id	if x is a character/factor vector, doc_id can be used to specify document ids. This has to be a vector of the same length as x
meta	A data.frame with document meta information (e.g., date, source). The rows of the data.frame need to match the values of x
split_sentences	Logical. If TRUE, the sentence number of tokens is also computed.
max_sentences	An integer. Limits the number of sentences per document to the specified number. If set when split_sentences == FALSE, split_sentences will be set to TRUE.
max_tokens	An integer. Limits the number of tokens per document to the specified number
verbose	If TRUE, report progress
text_columns	if x is a data.frame, this specifies the column(s) that contains text. The texts are paste together in the order specified here.
doc_column	If x is a data.frame, this specifies the column with the document ids.

Examples

```
tc = create_tcorpus(c('Text one first sentence. Text one second sentence', 'Text two'))
tc$get()

tc = create_tcorpus(c('Text one first sentence. Text one second sentence', 'Text two'),
                    split_sentences = TRUE)
tc$get()

## with meta (easier to S3 method for data.frame)
meta = data.frame(doc_id = c(1,2), source = c('a','b'))
tc = create_tcorpus(c('Text one first sentence. Text one second sentence', 'Text two'),
                    split_sentences = TRUE,
                    doc_id = c(1,2),
                    meta = meta)
tc
## It makes little sense to have full texts as factors, but it tends to happen.
## The create_tcorpus S3 method for factors is essentially identical to the
##  method for a character vector.
text = factor(c('Text one first sentence', 'Text one second sentence'))
```

```
tc = create_tcorpus(text)
tc$get()
d = data.frame(text = c('Text one first sentence. Text one second sentence.',
               'Text two', 'Text three'),
               date = c('2010-01-01','2010-01-01','2012-01-01'),
               source = c('A','B','B'))

tc = create_tcorpus(d, split_sentences = TRUE)
tc
tc$get()

## use multiple text columns
d$headline = c('Head one', 'Head two', 'Head three')
## use custom doc_id
d$doc_id = c('#1', '#2', '#3')

tc = create_tcorpus(d, text_columns = c('headline','text'), doc_column = 'doc_id',
                    split_sentences = TRUE)
tc
tc$get()

## (note that text from different columns is pasted together with a double newline in between)
tc$read_text(doc_id = '#1')
```

`docfreq_filter` *Support function for subset method*

Description

Support function to enable subsetting by document frequency stats of a given feature. Should only be used within the tCorpus subset method, or any tCorpus method that supports a subset argument.

Usage

```
docfreq_filter(x, min = -Inf, max = Inf, top = NULL, bottom = NULL,
  doc_id = parent.frame()$doc_id)
```

Arguments

`x`	the name of the feature column. Can be given as a call or a string.
`min`	A number, setting the minimum document frequency value
`max`	A number, setting the maximum document frequency value
`top`	A number. If given, only the top x features with the highest document frequency are TRUE
`bottom`	A number. If given, only the bottom x features with the highest document frequency are TRUE
`doc_id`	Added for reference, but should not be used. Automatically takes doc_id from tCorpus if the docfreq_filter function is used within the subset method.

Examples

```
tc = create_tcorpus(c('a a a b b', 'a a c c'))

tc$get()
tc$subset(subset = docfreq_filter(token, min=2))
tc$get()
```

`dtm_compare` *Compare two document term matrices*

Description

Compare two document term matrices

Usage

```
dtm_compare(dtm.x, dtm.y = NULL, smooth = 0.1, min_ratio = NULL,
  min_chi2 = NULL, select_rows = NULL, yates_cor = c("auto", "yes", "no"),
  x_is_subset = F, what = c("freq", "docfreq", "cooccurrence"))
```

Arguments

`dtm.x`	the main document-term matrix
`dtm.y`	the 'reference' document-term matrix
`smooth`	the smoothing parameter for Laplace smoothing.
`min_ratio`	threshold for the ratio value, which is the ratio of the relative frequency of a term in dtm.x and dtm.y
`min_chi2`	threshold for the chi^2 value
`select_rows`	Alternative to using dtm.y. Has to be a vector with rownames, by which
`yates_cor`	mode for using yates correctsion in the chi^2 calculation. Can be turned on ("yes") or off ("no"), or set to "auto", in which case cochrans rule is used to determine whether yates' correction is used.
`x_is_subset`	Specify whether dtm.x is a subset of dtm.y. In this case, the term frequencies of dtm.x will be subtracted from the term frequencies in dtm.y
`what`	choose whether to compare the frequency ("freq") of terms, or the document frequency ("docfreq"). This also affects how chi^2 is calculated, comparing either freq relative to vocabulary size or docfreq relative to corpus size (N)

Value

A data frame with rows corresponding to the terms in dtm and the statistics in the columns

dtm_wordcloud *Plot a word cloud from a dtm*

Description

Compute the term frequencies for the dtm and plot a word cloud with the top n topics You can either supply a document-term matrix or provide terms and freqs directly (in which case this is an alias for wordcloud::wordcloud with sensible defaults)

Usage

```
dtm_wordcloud(dtm = NULL, nterms = 100, freq.fun = NULL, terms = NULL,
  freqs = NULL, scale = c(6, 0.5), min.freq = 1, rot.per = 0.15, ...)
```

Arguments

dtm	the document-term matrix
nterms	the amount of words to plot (default 100)
freq.fun	if given, will be applied to the frequenies (e.g. sqrt)
terms	the terms to plot, ignored if dtm is given
freqs	the frequencies to plot, ignored if dtm is given
scale	the scale to plot (see wordcloud::wordcloud)
min.freq	the minimum frquency to include (see wordcloud::wordcloud)
rot.per	the percentage of vertical words (see wordcloud::wordcloud)
...	other arguments passed to wordcloud::wordcloud

Examples

```
## create DTM
tc = create_tcorpus(sotu_texts[1:100,], doc_column = 'id')
tc$preprocess('token', 'feature', remove_stopwords = TRUE)
dtm = tc$dtm('feature')

## Not run:
dtm_wordcloud(dtm, nterms = 20)

## or without a DTM
dtm_wordcloud(terms = c('in','the','cloud'), freqs = c(2,5,10))

## End(Not run)
```

ego_semnet *Create an ego network*

Description

Create an ego network from an igraph object.

Usage

```
ego_semnet(g, vertex_names, depth = 1, only_filter_vertices = T,
  weight_attr = "weight", min_weight = NULL, top_edges = NULL,
  max_edges_level = NULL, directed = c("out", "in"))
```

Arguments

g	an igraph object
vertex_names	a character string with the names of the ego vertices/nodes
depth	the number of degrees from the ego vertices/nodes that are included. 1 means that only the direct neighbours are included
only_filter_vertices	if True, the algorithm will only filter out vertices/nodes that are not in the ego network. If False (default) then it also filters out the edges.
weight_attr	the name of the edge attribute. if NA, no weight is used, and min_weight and top_edges are ignored
min_weight	a number indicating the minimum weight
top_edges	for each vertex within the given depth, only keep the top n edges with the strongest edge weight. Can also be a vector of the same length as the depth value, in which case a different value is used at each level: first value for level 1, second value for level 2, etc.
max_edges_level	the maximum number of edges to be added at each level of depth.
directed	if the network is directed, specify whether 'out' degrees or 'in' degrees are used

Details

The function is similar to the ego function in igraph, but with some notable differences. Firstly, if multiple vertex_names are given, the ego network for both is given in 1 network (whereas igraph creates a list of networks). Secondly, the min_weight and top_edges parameters can be used to focus on the strongest edges.

Examples

```
tc = create_tcorpus(c('a b c', 'd e f', 'a d'))
g = tc$semnet('token')

igraph::get.data.frame(g)
## Not run: plot_semnet(g)

## only keep nodes directly connected to given node
g_ego = ego_semnet(g, 'e')
igraph::get.data.frame(g_ego)
## Not run: plot_semnet(g_ego)

## only keep edges directly connected to given node
g_ego = ego_semnet(g, 'e', only_filter_vertices = FALSE)
igraph::get.data.frame(g_ego)
## Not run: plot_semnet(g_ego)

## only keep nodes connected to given node with a specified degree (i.e. distance)
g_ego = ego_semnet(g, 'e', depth = 2)
igraph::get.data.frame(g_ego)
## Not run: plot_semnet(g_ego)
```

---

freq_filter *Support function for subset method*

Description

Support function to enable subsetting by frequency stats of a given feature. Should only be used within the tCorpus subset method, or any tCorpus method that supports a subset argument.

Usage

```
freq_filter(x, min = -Inf, max = Inf, top = NULL, bottom = NULL)
```

Arguments

x	the name of the feature column. Can be given as a call or a string.
min	A number, setting the minimum frequency value
max	A number, setting the maximum frequency value
top	A number. If given, only the top x features with the highest frequency are TRUE
bottom	A number. If given, only the bottom x features with the highest frequency are TRUE

Examples

```
tc = create_tcorpus(c('a a a b b'))

tc$get()
tc$subset(subset = freq_filter(token, min=3))
tc$get()
```

`get_global_i` *Compute global feature positions*

Description

Features are given global ids, with an added distance (max_window_size) between contexts (e.g., documents, sentences). This way, the distance of features can be calculated across multiple contexts using a single vector

Usage

```
get_global_i(tc, context_level = c("document", "sentence"),
  max_window_size = 200)
```

Arguments

`tc`	tCorpus object
`context_level`	either 'document' or 'sentence'
`max_window_size`	Determines the size of the gap between documents. Called max_window_size because this gap determines what the maximum window size is for non-overlapping windows between documents

Value

a tCorpus object

`get_stopwords` *Get a character vector of stopwords*

Description

Get a character vector of stopwords

Usage

```
get_stopwords(lang)
```

Arguments

lang	The language. Current options are: "danish", "dutch", "english", "finnish", "french", "german", "hungarian", "italian", "norwegian", "portuguese", "romanian", "russian", "spanish" and "swedish"

Value

A character vector containing stopwords

Examples

```
en_stop = get_stopwords('english')
nl_stop = get_stopwords('dutch')
ge_stop = get_stopwords('german')

head(en_stop)
head(nl_stop)
head(ge_stop)
```

---

laplace *Laplace (i.e. add constant) smoothing*

Description

Laplace (i.e. add constant) smoothing

Usage

```
laplace(freq, add = 0.5)
```

Arguments

freq	A numeric vector of term frequencies (integers).
add	The added value

Value

A numeric vector with the smoothed term proportions

Examples

```
laplace(c(0,0,1,1,1,2,2,2,3,3,4,7,10))
```

merge_tcorpora *Merge tCorpus objects*

Description

Create one tcorpus based on multiple tcorpus objects

Usage

```
merge_tcorpora(..., keep_data = c("intersect", "all"),
  keep_meta = c("intersect", "all"), if_duplicate = c("stop", "rename",
  "drop"), duplicate_tag = "#D")
```

Arguments

...	tCorpus objects, or a list with tcorpus objects
keep_data	if 'intersect', then only the token data columns that occur in all tCorpurs objects are kept
keep_meta	if 'intersect', then only the document meta columns that occur in all tCorpurs objects are kept
if_duplicate	determine behaviour if there are duplicate doc_ids across tcorpora. By default, this yields an error, but you can set it to "rename" to change the names of duplicates (which makes sense of only the doc_ids are duplicate, but not the actual content), or "drop" to ignore duplicates, keeping only the first unique occurence.
duplicate_tag	a character string. if if_duplicates is "rename", this tag is added to the document id. (this is repeated till no duplicates remain)

Value

a tCorpus object

Examples

```
tc1 = create_tcorpus(sotu_texts[1:10,], doc_column = 'id')
tc2 = create_tcorpus(sotu_texts[11:20,], doc_column = 'id')
tc = merge_tcorpora(tc1, tc2)
tc$n_meta

#### duplicate handling ####
tc1 = create_tcorpus(sotu_texts[1:10,], doc_column = 'id')
tc2 = create_tcorpus(sotu_texts[6:15,], doc_column = 'id')

## duplicate error
## Not run: tc = merge_tcorpora(tc1,tc2)

## with "rename", has 20 documents of which 5 duplicates
tc = merge_tcorpora(tc1,tc2, if_duplicate = 'rename')
```

```
tc$n_meta
sum(grepl('#D', tc$get_meta()$doc_id))

## with "drop", has 15 documents without duplicates
tc = merge_tcorpora(tc1,tc2, if_duplicate = 'drop')
tc$n_meta
mean(grepl('#D', tc$get_meta()$doc_id))
```

`plot.vocabularyComparison` *visualize vocabularyComparison*

Description

visualize vocabularyComparison

Usage

```
## S3 method for class 'vocabularyComparison'
plot(x, n = 25, mode = c("both", "ratio_x",
  "ratio_y"), balance = T, size = c("chi2", "freq", "ratio"), ...)
```

Arguments

x	a vocabularyComparison object, created with the compare_corpus or compare_subset method
n	the number of words in the plot
mode	use "both" to plot both overrepresented and underrepresented words using the plot_words function. Use "ratio_x" or "ratio_y" to only plot overrepresented or underrepresented words using dtm_wordcloud
balance	if TRUE, get an equal amount of terms on the left (underrepresented) and right (overrepresented) side. If FALSE, the top chi words are used, regardless of ratio.
size	use "freq", "chi2" or "ratio" for determining the size of words
...	additional arguments passed to plot_words ("both" mode) or dtm_wordcloud (ratio modes)

Examples

```
## as example, compare SOTU paragraphs about taxes to rest
tc = create_tcorpus(sotu_texts[1:100,], doc_column = 'id')
comp = tc$compare_subset('token', query_x = 'tax*')

## Not run:
plot(comp, balance=T)
plot(comp, mode = 'ratio_x')
plot(comp, mode = 'ratio_y')

## End(Not run)
```

plot_semnet *Visualize a semnet network*

Description

plot_semnet is a wrapper for the plot.igraph() function optimized for plotting a semantic network of the "semnet" class.

Usage

```
plot_semnet(g, weight_attr = "weight", min_weight = NA,
  delete_isolates = F, vertexsize_attr = "freq", vertexsize_coef = 1,
  vertexcolor_attr = NA, edgewidth_coef = 1, max_backbone_alpha = NA,
  labelsize_coef = 1, labelspace_coef = 1.1, reduce_labeloverlap = F,
  redo_layout = F, return_graph = T, vertex.label.dist = 0.25,
  layout_fun = igraph::layout_with_fr, ...)
```

Arguments

g	A network in the igraph format. Specifically designed for the output of coOccurenceNetwork() and windowedCoOccurenceNetwork()
weight_attr	The name of the weight attribute. Default is 'weight'
min_weight	The minimum weight. All edges with a lower weight are dropped
delete_isolates	If TRUE, isolate vertices (also after applying min_weight) are dropped
vertexsize_attr	a character string indicating a vertex attribute that represents size. Default is 'freq', which is created in the coOccurenceNetwork functions to indicate the number of times a token occured.
vertexsize_coef	a coefficient for changing the vertex size.
vertexcolor_attr	a character string indicating a vertex attribute that represents color. The attribute can also be a numeric value (e.g., a cluster membership) in which case colors are assigned to numbers. If no (valid) color attribute is given, vertex color are based on undirected fastgreedy.community() clustering.
edgewidth_coef	a coefficient for changing the edge width
max_backbone_alpha	If g has an edge attribute named alpha (added if backbone extraction is used), this specifies the maximum alpha value.
labelsize_coef	a coefficient for increasing or decreasing the size of the vertexlabel.
labelspace_coef	a coefficient that roughly determines the minimal distance between vertex labels, based on the size of labels. Only used if reduce_labeloverlap is TRUE.

reduce_labeloverlap
: if TRUE, an algorithm is used to reduce overlap as best as possible.

redo_layout
: If TRUE, a new layout will be calculated using layout_with_fr(). If g does not have a layout attribute (g$layout), a new layout is automatically calculated.

return_graph
: if TRUE, plot_semnet() also returns the graph object with the attributes and layout as shown in the plot.

vertex.label.dist
: The distance of the label to the center of the vertex

layout_fun
: The igraph layout function that is used.

...
: additional arguments are passed on to plot.igraph()

Details

Before plotting the network, the set_network_attributes() function is used to set pretty defaults for plotting. Optionally, reduce_labeloverlap can be used to prevent labeloverlap (as much as possible).

Value

Plots a network, and returns the network object if return_graph is TRUE.

Examples

```
tc = create_tcorpus(sotu_texts, doc_column = 'id')
tc$preprocess('token','feature', remove_stopwords = TRUE, use_stemming = TRUE, min_docfreq=10)
## Not run:
g = tc$semnet_window('feature', window.size = 10)
g = backbone_filter(g, max_vertices = 100)
plot_semnet(g)

## End(Not run)
```

---

plot_words *Plot a wordcloud with words ordered and coloured according to a dimension (x)*

Description

Plot a wordcloud with words ordered and coloured according to a dimension (x)

Usage

```
plot_words(x, y = NULL, words, wordfreq = rep(1, length(x)), xlab = "",
  ylab = "", yaxt = "n", scale = 2, random.y = T, xlim = NULL,
  ylim = NULL, ...)
```

Arguments

x	The (approximate) x positions of the words
y	The (approximate) y positions of the words
words	A character vector with the words to plot
wordfreq	The frequency of the words, defaulting to 1
xlab	Label of the x axis
ylab	Label of the y axis
yaxt	see par documentation
scale	Maximum size to scale the wordsize
random.y	if TRUE, the y position of words is random, otherwise it represents the word frequency.
xlim	Starting value of x axis
ylim	Starting value of y axis
...	additional parameters passed to the plot function

Value

nothing

Examples

```
x = c(-10, -5, 3, 5)
y = c(0, 2, 5, 10)
words = c('words', 'where', 'you', 'like')

## Not run:
plot_words(x,y,words, c(1,2,3,4))

## End(Not run)
```

---

preprocess_tokens *Preprocess tokens in a character vector*

Description

Preprocess tokens in a character vector

Usage

```
preprocess_tokens(x, context = NULL, language = "english",
  use_stemming = F, lowercase = T, ngrams = 1, replace_whitespace = T,
  as_ascii = F, remove_punctuation = T, remove_stopwords = F,
  min_freq = NULL, min_docfreq = NULL)
```

Arguments

x	A character vector in which each element is a token (i.e. a tokenized text)
context	Optionally, a character vector of the same length as x, specifying the context of token (e.g., document, sentence). Has to be given if ngram > 1
language	The language used for stemming and removing stopwords
use_stemming	Logical, use stemming. (Make sure the specify the right language!)
lowercase	Logical, make token lowercase
ngrams	A number, specifying the number of tokens per ngram. Default is unigrams (1).
replace_whitespace	Logical. If TRUE, all whitespace is replaced by underscores
as_ascii	Logical. If TRUE, tokens will be forced to ascii
remove_punctuation	Logical. if TRUE, punctuation is removed
remove_stopwords	Logical. If TRUE, stopwords are removed (Make sure to specify the right language!)
min_freq	an integer, specifying minimum token frequency.
min_docfreq	an integer, specifying minimum document frequency.

Examples

```
tokens = c('I', 'am', 'a', 'SHORT', 'example', 'sentence', '!')

## default is lowercase without punctuation
preprocess_tokens(tokens)

## optionally, delete stopwords, perform stemming, and make ngrams
preprocess_tokens(tokens, remove_stopwords = TRUE, use_stemming = TRUE)
preprocess_tokens(tokens, context = NA, ngrams = 3)
```

---

print.contextHits *S3 print for contextHits class*

Description

S3 print for contextHits class

Usage

```
## S3 method for class 'contextHits'
print(x, ...)
```

Arguments

x	a contextHits object, as returned by tCorpus$search_contexts
...	not used

Examples

```
text = c('A B C', 'D E F. G H I', 'A D', 'GGG')
tc = create_tcorpus(text, doc_id = c('a','b','c','d'), split_sentences = TRUE)
hits = tc$search_contexts(c('query label# A AND B', 'second query# (A AND Q) OR ("D E") OR I'))

hits
```

---

print.featureHits *S3 print for featureHits class*

Description

S3 print for featureHits class

Usage

```
## S3 method for class 'featureHits'
print(x, ...)
```

Arguments

x	a featureHits object, as returned by tCorpus$search_features
...	not used

Examples

```
text = c('A B C', 'D E F. G H I', 'A D', 'GGG')
tc = create_tcorpus(text, doc_id = c('a','b','c','d'), split_sentences = TRUE)
hits = tc$search_features(c('query label# A AND B', 'second query# (A AND Q) OR ("D E") OR I'))

hits
```

print.tCorpus *S3 print for tCorpus class*

Description

S3 print for tCorpus class

Usage

```
## S3 method for class 'tCorpus'
print(x, ...)
```

Arguments

x	a tCorpus object
...	not used

Examples

```
tc = create_tcorpus(c('First text', 'Second text'))
print(tc)
```

refresh_tcorpus *Refresh a tCorpus object using the current version of corpustools*

Description

As an R6 class, tCorpus contains its methods within the class object (i.e. itself). Therefore, if you use a new version of corpustools with an older tCorpus object (e.g., stored as a .rds. file), then the methods are not automatically updated. You can then use refresh_tcorpus() to reinitialize the tCorpus object with the current version of corpustools.

Usage

```
refresh_tcorpus(tc)
```

Arguments

tc	a tCorpus object

Value

a tCorpus object

Examples

```
tc = create_tcorpus(c('First text', 'Second text'))
refresh_tcorpus(tc)
```

set_network_attributes *Set some default network attributes for pretty plotting*

Description

The purpose of this function is to create some default network attribute options to plot networks in a nice and insightfull way.

Usage

```
set_network_attributes(g, size_attribute = "freq", color_attribute = NA,
  redo_layout = F, edgewidth_coef = 1,
  layout_fun = igraph::layout_with_fr)
```

Arguments

g	A graph in the Igraph format.
size_attribute	the name of the vertex attribute to be used to set the size of nodes
color_attribute	the name of the attribute that is used to select the color
redo_layout	if TRUE, force new layout if layout already exists as a graph attribute
edgewidth_coef	A coefficient for changing the edge width
layout_fun	THe igraph layout function used

Value

a network in the Igraph format

Examples

```
tc = create_tcorpus(c('A B C', 'B C', 'B D'))
g = tc$semnet('token')

igraph::get.edge.attribute(g)
igraph::get.vertex.attribute(g)
## Not run: plot(g)
g = set_network_attributes(g, size_attribute = 'freq')
igraph::get.edge.attribute(g)
igraph::get.vertex.attribute(g)
## Not run: plot(g)
```

sgt *Simple Good Turing smoothing*

Description

Implementation of the Simple Good Turing smoothing proposed in: Gale, W. A., \& Sampson, G. (1995). Good turing frequency estimation without tears. Journal of Quantitative Linguistics, 2(3), 217-237.

Usage

```
sgt(freq)
```

Arguments

freq	A numeric vector of term frequencies (integers).

Value

A numeric vector with the smoothed term proportions

sotu_texts *State of the Union addresses*

Description

State of the Union addresses

Usage

```
data(sotu_texts)
```

Format

data.frame

stopwords_list *Basic stopword lists*

Description

Basic stopword lists

Usage

```
data(stopwords_list)
```

Format

A named list, with names matching the languages used by SnowballC

summary.contextHits *S3 summary for contextHits class*

Description

S3 summary for contextHits class

Usage

```
## S3 method for class 'contextHits'
summary(object, ...)
```

Arguments

object	a contextHits object, as returned by tCorpus$search_contexts
...	not used

Examples

```
text = c('A B C', 'D E F. G H I', 'A D', 'GGG')
tc = create_tcorpus(text, doc_id = c('a','b','c','d'), split_sentences = TRUE)
hits = tc$search_contexts(c('query label# A AND B', 'second query# (A AND Q) OR ("D E") OR I'))

summary(hits)
```

summary.featureHits *S3 summary for featureHits class*

Description

S3 summary for featureHits class

Usage

```
## S3 method for class 'featureHits'
summary(object, ...)
```

Arguments

object	a featureHits object, as returned by tCorpus$search_features
...	not used

Examples

```
text = c('A B C', 'D E F. G H I', 'A D', 'GGG')
tc = create_tcorpus(text, doc_id = c('a','b','c','d'), split_sentences = TRUE)
hits = tc$search_features(c('query label# A AND B', 'second query# (A AND Q) OR ("D E") OR I'))

summary(hits)
```

summary.tCorpus *Summary of a tCorpus object*

Description

Summary of a tCorpus object

Usage

```
## S3 method for class 'tCorpus'
summary(object, ...)
```

Arguments

object	A tCorpus object
...	not used

Examples

```
tc = create_tcorpus(c('First text', 'Second text'))
summary(tc)
```

`tCorpus` *tCorpus: a corpus class for tokenized texts*

Description

tCorpus: a corpus class for tokenized texts

Working with the tCorpus

The primary goal of the tCorpus is to facilitate various corpus analysis techniques. The documentation for currently implemented techniques can be reached through the following links.

Create a tCorpus	Functions for creating a tCorpus object
Manage tCorpus data	Methods for viewing, modifying and subsetting tCorpus data
Features	Preprocessing, subsetting and analyzing features
Using search strings	Use Boolean queries to analyze the tCorpus
Co-occurrence networks	Feature co-occurrence based semantic network analysis
Corpus comparison	Compare corpora
Topic modeling	Create and visualize topic models
Document similarity	Calculate document similarity

`tCorpus$code_features` *Code features in a tCorpus based on a search string*

Description

Add a column to the token data that contains a code (the query label) for tokens that match the query (see tCorpus$search_features).

Arguments

`query`	A character string that is a query. See search_features for documentation of the query language.
`code`	The code given to the tokens that match the query (usefull when looking for multiple queries). Can also put code label in query with # (see details)
`feature`	The name of the feature column within which to search.
`column`	The name of the column that is added to the data
`add_column`	list of name-value pairs, used to add additional columns. The name will become the column name, and the value should be a vector of the same length as the query vector.
`context_level`	Select whether the queries should occur within while "documents" or specific "sentences".

keep_longest	If TRUE, then overlapping in case of overlapping queries strings in unique_hits mode, the query with the most separate terms is kept. For example, in the text "mr. Bob Smith", the query [smith OR "bob smith"] would match "Bob" and "Smith". If keep_longest is FALSE, the match that is used is determined by the order in the query itself. The same query would then match only "Smith".
as_ascii	if TRUE, perform search in ascii.
verbose	If TRUE, progress messages will be printed
...	alternative way to specify name-value pairs for adding additional columns

Usage

R6 method for class tCorpus. Use as tc$method (where tc is a tCorpus object).

```
code_features(query, code=NULL, feature='token', column='code', ...)
```

Examples

```
tc = create_tcorpus('Anna and Bob are secretive')

tc$code_features(c("actors# anna bob", "associations# secretive"))
tc$get()
```

---

tCorpus$compare_corpus

Compare tCorpus vocabulary to that of another (reference) tCorpus

---

Description

Compare tCorpus vocabulary to that of another (reference) tCorpus

Arguments

tc_y	the reference tCorpus
feature	the column name of the feature that is to be compared
smooth	Laplace smoothing is used for the calculation of the ratio of the relative term frequency. Here you can set the added value.
min_ratio	threshold for the ratio value, which is the ratio of the relative frequency of a term in dtm.x and dtm.y
min_chi2	threshold for the chi^2 value
yates_cor	mode for using yates correctsion in the chi^2 calculation. Can be turned on ("yes") or off ("no"), or set to "auto", in which case cochrans rule is used to determine whether yates' correction is used.
is_subset	Specify whether tc is a subset of tc_y. In this case, the term frequencies of tc will be subtracted from the term frequencies in tc_y
what	choose whether to compare the frequency ("freq") of terms, or the document frequency ("docfreq"). This also affects how chi^2 is calculated, comparing either freq relative to vocabulary size or docfreq relative to corpus size (N)

Value

A vocabularyComparison object

Usage

R6 method for class tCorpus. Use as tc$method (where tc is a tCorpus object).

```
compare_corpus(tc_y, feature, smooth=0.1, min_ratio=NULL, min_chi2=NULL, is_subset=F, yates_cor=c('au
```

Examples

```
tc = create_tcorpus(sotu_texts, doc_column = 'id')

tc$preprocess('token', 'feature', remove_stopwords = TRUE, use_stemming = TRUE)

obama = tc$subset_meta(president == 'Barack Obama', copy=TRUE)
bush = tc$subset_meta(president == 'George W. Bush', copy=TRUE)

comp = obama$compare_corpus(bush, 'feature')
comp = comp[order(-comp$chi),]
head(comp)
## Not run:
plot(comp)

## End(Not run)
```

---

tCorpus$compare_documents

Calculate the similarity of documents

Description

Calculate the similarity of documents

Arguments

feature	the column name of the feature that is to be used for the comparison.
date_col	a date with time in POSIXct. If given together with hour_window, only documents within the given hour_window will be compared.
hour_window	an integer. If given together with date_col, only documents within the given hour_window will be compared.
measure	the similarity measure. Currently supports cosine similarity (symmetric) and overlap_pct (asymmetric)
weight	a weighting scheme for the document-term matrix. Default is term-frequency inverse document frequency with normalized rows (document length).
ngrams	an integer. If given, ngrams of this length are used

from_subset	An expression to select a subset. If given, only this subset will be compared to other documents
to_subset	An expression to select a subset. If given, documents are only compared to this subset

Usage

R6 method for class tCorpus. Use as tc$method (where tc is a tCorpus object).

```
compare_documents(feature='token', date_col=NULL, hour_window=NULL, measure=c('cosine','overlap_pct'
```

Examples

```
d = data.frame(text = c('a b c d e',
                        'e f g h i j k',
                        'a b c'),
               date = c('2010-01-01','2010-01-01','2012-01-01'))
tc = create_tcorpus(d)

g = tc$compare_documents()
igraph::get.data.frame(g)

g = tc$compare_documents(measure = 'overlap_pct')
igraph::get.data.frame(g)

g = tc$compare_documents(date_col = 'date', hour_window = c(0,36))
igraph::get.data.frame(g)
```

---

tCorpus$compare_subset

Compare vocabulary of a subset of a tCorpus to the rest of the tCorpus

---

Description

Compare vocabulary of a subset of a tCorpus to the rest of the tCorpus

Arguments

feature	the column name of the feature that is to be compared
subset_x	an expression to subset the tCorpus. The vocabulary of the subset will be compared to the rest of the tCorpus
subset_meta_x	like subset_x, but using using the meta data
query_x	like subset_x, but using a query search to select documents (see tCorpus$search_contexts)
query_feature	if query_x is used, the column name of the feature used in the query search.
smooth	Laplace smoothing is used for the calculation of the ratio of the relative term frequency. Here you can set the added value.

min_ratio	threshold for the ratio value, which is the ratio of the relative frequency of a term in dtm.x and dtm.y
min_chi2	threshold for the chi^2 value
yates_cor	mode for using yates correctsion in the chi^2 calculation. Can be turned on ("yes") or off ("no"), or set to "auto", in which case cochrans rule is used to determine whether yates' correction is used.
what	choose whether to compare the frequency ("freq") of terms, or the document frequency ("docfreq"). This also affects how chi^2 is calculated, comparing either freq relative to vocabulary size or docfreq relative to corpus size (N)

Value

A vocabularyComparison object

Usage

R6 method for class tCorpus. Use as tc$method (where tc is a tCorpus object).

```
compare_subset(feature, subset_x=NULL, subset_meta_x=NULL, query_x=NULL, query_feature='token', smoo
```

Examples

```
tc = create_tcorpus(sotu_texts, doc_column = 'id')

tc$preprocess('token', 'feature', remove_stopwords = TRUE, use_stemming = TRUE)

comp = tc$compare_subset('feature', subset_meta_x = president == 'Barack Obama')
comp = comp[order(-comp$chi),]
head(comp)
## Not run:
plot(comp)

## End(Not run)

comp = tc$compare_subset('feature', query_x = 'terroris*')
comp = comp[order(-comp$chi),]
head(comp, 10)
```

---

tCorpus$context *Get a context vector*

Description

Depending on the purpose, the context of an analysis can be the document level or sentence level. the tCorpus$context() method offers a convenient way to get the context id of tokens for different settings.

Arguments

context_level	Select whether the context is document or sentence level
with_labels	Return context as only ids (numeric, starting at 1) or with labels (factor)

Usage

R6 method for class tCorpus. Use as tc$method (where tc is a tCorpus object).

```
data(context_level = c('document','sentence'), with_labels = T)
```

Examples

```
tc <- create_tcorpus(c('Text one first sentence. Text one second sentence', 'Text two'),
                     split_sentences = TRUE)

doc <- tc$context() ## default context is doc_id (document level)
doc

sent <- tc$context('sentence') ## can specify sentence level
sent
```

---

tCorpus$deduplicate *Deduplicate documents*

Description

Deduplicate documents based on similarity scores. Can be used to filter out identical documents, but also similar documents.

Arguments

feature	the column name of the feature that is to be used for the comparison.
date_col	The column name for a column with a date vector (in POSIXct). If given together with hour_window, only documents within the given hour_window will be compared.
meta_cols	a vector with names for columns in the meta data. If given, documents are only considered duplicates if the values of these columns are identical (in addition to having a high similarity score)
hour_window	an integer. If given together with date_col, only documents within the given hour_window will be compared.
min_docfreq	a minimum document frequency for features. This is mostly to lighten computational load. Default is 2, because terms that occur once cannot overlap across documents
min_docfreq	a maximum document frequency percentage for features. High frequency terms contain little information for identifying duplicates. Default is 0.5 (i.e. terms that occur in more than 50 percent of documents are ignored),

measure	the similarity measure. Currently supports cosine similarity (symmetric) and overlap_pct (asymmetric)
similarity	the similarity threshold used to determine whether two documents are duplicates. Default is 1, meaning 100 percent identical.
keep	select either 'first', 'last' or 'random'. Determines which document of duplicates to delete. If a date is given, 'first' and 'last' specify whether the earliest or latest document is kept.
weight	a weighting scheme for the document-term matrix. Default is term-frequency inverse document frequency with normalized rows (document length).
ngrams	an integer. If given, ngrams of this length are used
print_deduplicates	if TRUE, print ids of duplicates that are deleted
copy	If TRUE, the method returns a new tCorpus object instead of deduplicating the current one by reference.

Details

Note that deduplication occurs by reference (tCorpus_modify_by_reference) unless copy is set to TRUE.

Usage

R6 method for class tCorpus. Use as tc$method (where tc is a tCorpus object).

```
deduplicate(feature='token', date_col=NULL, meta_cols=NULL, hour_window=NULL, min_docfreq=2, max_doc
```

Examples

```
d = data.frame(text = c('a b c d e',
                        'e f g h i j k',
                        'a b c'),
               date = c('2010-01-01','2010-01-01','2012-01-01'))
tc = create_tcorpus(d)

tc$get_meta()
dedup = tc$deduplicate(feature='token', date_col = 'date', similarity = 0.8, copy=TRUE)
dedup$get_meta()

dedup = tc$deduplicate(feature='token', date_col = 'date', similarity = 0.8, keep = 'last',
                       copy=TRUE)
dedup$get_meta()
```

tCorpus$delete_columns *Delete column from the data and meta data*

Description

Delete column from the data and meta data

Arguments

cnames	the names of the columns to delete

Usage

R6 method for class tCorpus. Use as tc$method (where tc is a tCorpus object).

```
delete_columns(cnames)


delete_meta_columns(cnames)
```

Examples

```
d = data.frame(text = c('Text one','Text two','Text three'),
               date = c('2010-01-01','2010-01-01','2012-01-01'))
tc = create_tcorpus(d)

tc$get()
tc$delete_columns('token')
tc$get()

tc$get_meta()
tc$delete_meta_columns('date')
tc$get_meta()
```

tCorpus$dtm *Create a document term matrix*

Description

Create a document term matrix

Arguments

feature	The name of the feature column
context_level	Select whether the rows of the dtm should represent "documents" or "sentences".
weight	Select the weighting scheme for the DTM. Currently supports term frequency (termfreq), document frequency (docfreq), term frequency inverse document frequency (tfidf) and tfidf with normalized document vectors.
drop_empty_terms	If True, tokens that do not occur (i.e. column where sum is 0) are ignored.
form	The output format. Default is a sparse matrix in the dgTMatrix class from the Matrix package. Alternatives are tm_dtm for a DocumentTermMatrix in the tm package format or quanteda_dfm for the document feature matrix from the quanteda package.
subset_tokens	A subset call to select which rows to use in the DTM
subset_meta	A subset call for the meta data, to select which documents to use in the DTM
context	Instead of using the document or sentence context, an custom context can be specified. Has to be a vector of the same length as the number of tokens, that serves as the index column. Each unique value will be a row in the DTM.
context_labels	If False, the DTM will not be given rownames
feature_labels	If False, the DTM will not be given column names
ngrams	Optionally, use ngrams instead of individual tokens. This is more memory efficient than first creating an ngram feature in the tCorpus.
ngram_before_subset	If a subset is used, ngrams can be made before the subset, in which case an ngram can contain tokens that have been filtered out after the subset. Alternatively, if ngrams are made after the subset, ngrams will span over the gaps of tokens that are filtered out.

Usage

R6 method for class tCorpus. Use as tc$method (where tc is a tCorpus object).

```
dtm(feature, context_level=c('document','sentence'), weight=c('termfreq','docfreq','tfidf','norm_tfi
```

Examples

```
tc = create_tcorpus(c("First text first sentence. First text first sentence.",
                "Second text first sentence"), doc_column = 'id', split_sentences = TRUE)

## Perform additional preprocessing on the 'token' column, and save as the 'feature' column
tc$preprocess('token', 'feature', remove_stopwords = TRUE, use_stemming = TRUE)
tc$get()

## default: regular sparse matrix, using the Matrix package
m = tc$dtm('feature')
class(m)
m
```

```
## alternatively, create quanteda ('quanteda_dfm') or tm ('tm_dtm') class for DTM
## Not run:
m = tc$dtm('feature', form = 'quanteda_dfm')
class(m)
m

## End(Not run)

## create DTM with sentences as rows (instead of documents)
m = tc$dtm('feature', context_level = 'sentence')
nrow(m)

## use weighting
m = tc$dtm('feature', weight = 'norm_tfidf')
```

`tCorpus$feature_associations`

Get common nearby terms given a feature query

Description

Get common nearby terms given a feature query

Arguments

`query`	A character string that is a query. See search_features for documentation of the query language.
`hits`	Alternatively, instead of giving a query, the results of tCorpus$search_features can be used.
`feature`	If keyword is used, the name of the feature column within which to search.
`window`	The size of the word window (i.e. the number of words next to the feature)
`n`	the top n of associated features
`min_freq`	Optionally, ignore features that occur less than min_freq times
`sort_by`	The value by which to sort the features
`subset`	A call (or character string of a call) as one would normally pass to subset.tCorpus. If given, the keyword has to occur within the subset. This is for instance usefull to only look in named entity POS tags when searching for people or organization. Note that the condition does not have to occur within the subset.
`subset_meta`	A call (or character string of a call) as one would normally pass to the subset_meta parameter of subset.tCorpus. If given, the keyword has to occur within the subset documents. This is for instance usefull to make queries date dependent. For example, in a longitudinal analysis of politicians, it is often required to take changing functions and/or party affiliations into account. This can be accomplished by using subset_meta = "date > xxx & date < xxx" (given that the appropriate date column exists in the meta data).

Usage

R6 method for class tCorpus. Use as tc$method (where tc is a tCorpus object).

```
feature_associations(query=NULL, hits=NULL, feature='token',
                             window=15,  n=25, min_freq=1, sort_by= c('chi2', 'ratio', 'freq'),
                                    subset=NULL, subset_meta=NULL
```

Examples

```
tc = create_tcorpus(sotu_texts, doc_column = 'id')

## directly from query
topf = tc$feature_associations('war')
head(topf, 20) ## frequent words close to "war"

## adjust window size
topf = tc$feature_associations('war', window = 5)
head(topf, 20) ## frequent words very close (five tokens) to "war"

## you can also first perform search_features, to get hits for (complex) queries
hits = tc$search_features('"war terror"~10')
topf = tc$feature_associations(hits = hits)
head(topf, 20) ## frequent words close to the combination of "war" and "terror" within 10 words
```

---

tCorpus$feature_stats *Feature statistics*

Description

Compute a number of useful statistics for features: term frequency, idf, etc.

Arguments

feature	The name of the feature column
sent_freq	If True, include sentence frequency (only if sentence information is available).

Usage

R6 method for class tCorpus. Use as tc$method (where tc is a tCorpus object).

```
feature_stats(feature, sent_freq=F)
```

Examples

```
tc = create_tcorpus(c('Text one first sentence. Text one second sentence', 'Text two'),
                    split_sentences = TRUE)

fs = tc$feature_stats('token')
head(fs)

fs = tc$feature_stats('token', context_level = 'sentence')
head(fs)
```

---

tCorpus$feature_subset

Filter features

Description

Similar to using tCorpus$subset, but instead of deleting rows it only sets rows for a specified feature to NA. This can be very convenient, because it enables only a selection of features to be used in an analysis (e.g. a topic model) but maintaining the context of the full article, so that results can be viewed in this context (e.g. a topic browser).

Just as in subset, it is easy to use objects and functions in the filter, including the special functions for using term frequency statistics (see documentation for tCorpus$subset).

Arguments

column	the column containing the feature to be used as the input
new_column	the column to save the filtered feature. Can be a new column or overwrite an existing one.
subset	logical expression indicating rows to keep in the tokens data. i.e. rows for which the logical expression is FALSE will be set to NA.

Usage

R6 method for class tCorpus. Use as tc$method (where tc is a tCorpus object).

```
feature_subset(column, new_column, subset)
```

Examples

```
tc = create_tcorpus('a a a a b b b c c')

tc$feature_subset('token', 'tokens_subset1', subset = token_id < 5)
tc$feature_subset('token', 'tokens_subset2', subset = freq_filter(token, min = 3))

tc$get()
```

`tCorpus$get` *Access the data from a tCorpus*

Description

Get the token and meta data.

Arguments

`columns`	character vector with the names of the columns
`keep_df`	if True, the output will be a data.table (or data.frame) even if it only contains 1 columns
`as.df`	if True, the output will be a regular data.frame instead of a data.table
`subset`	Optionally, only get a subset of rows (see tCorpus$subset method)
`doc_id`	A vector with document ids to select rows. Faster than subset, because it uses binary search. Cannot be used in combination with subset. If duplicate doc_ids are given, duplicate rows are returned.
`token_id`	A vector with token indices. Can only be used in pairs with doc_id. For example, if doc_id = c(1,1,1,2,2) and token_id = c(1,2,3,1,2), then the first three tokens of doc 1 and the first 2 tokens of doc 2 are returned. This is mainly usefull for fast (binary search) retrieval of specific tokens.
`safe_copy`	for advanced use. The get methods always return a copy of the data, even if the full data is returned (i.e. use get without parameters). This is to prevent accidental changes within tCorpus data (which can break it) if the returned data is modified by reference (see data.table documentation). If safe_copy is set to FALSE and get is called without parameters—tc$get(safe_copy=F))—then no copy is made, which is much faster and more memory efficient. Use this if you need speed and efficiency, but make sure not to change the output data.table by reference.

Usage

R6 active method for class tCorpus. Use as tc$method (where tc is a tCorpus object).

```
get(columns=NULL, keep_df=F, as.df=F, subset=NULL, doc_id=NULL, token_id=NULL, safe_copy=T)

get_meta(columns=NULL, keep_df=F, as.df=F, subset=NULL, doc_id=NULL, safe_copy=T)
```

Examples

```
d = data.frame(text = c('Text one first sentence. Text one second sentence', 'Text two'),
               medium = c('A','B'),
               date = c('2010-01-01','2010-02-01'),
               doc_id = c('D1','D2'))
tc = create_tcorpus(d, split_sentences = TRUE)
```

```
## get token data
tc$get()                      ## full data.table
tc$get(c('doc_id','token'))  ## data.table with selected columns
head(tc$get('doc_id'))       ## single column as vector
head(tc$get(as.df = TRUE))      ## return as regular data.frame

## get subset
tc$get(subset = token_id %in% 1:2)

## subset on keys using (fast) binary search
tc$get(doc_id = 'D1')              ## for doc_id
tc$get(doc_id = 'D1', token_id = 5) ## for doc_id / token pairs


##### use get for meta data with get_meta
tc$get_meta()

## option to repeat meta data to match tokens
tc$get_meta(per_token = TRUE) ## (note that first doc is repeated, and rows match tc$n)
```

`tCorpus$kwic` *Get keyword-in-context (KWIC) strings*

Description

Create a data.frame with keyword-in-context strings for given indices (i), search results (hits) or search strings (keyword).

Arguments

`hits`	results of feature search. see tCorpus$search_features.
`i`	instead of the hits argument, you can give the indices of features directly.
`query`	instead of using the hits or i arguments, a search string can be given directly. Note that this simply a convenient shorthand for first creating a hits object with tCorpus$search_features. If a query is given, then the ... argument is used to pass other arguments to tCorpus$search_features.
`code`	if 'i' or 'query' is used, the code argument can be used to add a code label. Should be a vector of the same length that gives the code for each i or query, or a vector of length 1 for a single label.
`ntokens`	an integers specifying the size of the context, i.e. the number of tokens left and right of the keyword.
`n`	a number, specifying the total number of hits
`nsample`	like n, but with a random sample of hits. If multiple codes are used, the sample is drawn for each code individually.
`output_feature`	the feature column that is used to make the KWIC.

context_level	Select the maxium context (document or sentence).
kw_tag	a character vector of length 2, that gives the symbols before (first value) and after (second value) the keyword in the KWIC string. Can for instance be used to prepare KWIC with format tags for highlighting.
...	See tCorpus$search_features for the query parameters

Usage

R6 method for class tCorpus. Use as tc$method (where tc is a tCorpus object).

```
kwic(hits = NULL, i = NULL, query = NULL, code = '',
     ntokens = 10, nsample = NA, output_feature = 'token',
     context_levels = c('document','sentence'),
     prettypaste = T, kw_tag = c('<','>'), ...)
```

Examples

```
tc = tokens_to_tcorpus(corenlp_tokens, sentence_col = 'sentence', token_id_col = 'id')

## look directly for a term (or complex query)
tc$kwic(query = 'love*')

## or, first perform a feature search, and then get the KWIC for the results
hits = tc$search_features('(john OR mark) AND mary AND love*', context_level = 'sentence')
tc$kwic(hits, context_level = 'sentence')
```

---

tCorpus$lda_fit *Estimate a LDA topic model*

Description

Estimate an LDA topic model using the LDA function from the topicmodels package. The parameters other than dtm are simply passed to the sampler but provide a workable default. See the description of that function for more information

Arguments

feature	the name of the feature columns
create_feature	optionally, add a feature column that indicates the topic to which a feature was assigned (in the last iteration). Has to be a character string, that will be the name of the new feature column
K	the number of clusters
num.iterations	the number of iterations
method	set method. see documentation for LDA function of the topicmodels package
alpha	the alpha parameter
eta	the eta parameter#'
burnin	The number of burnin iterations

Value

A fitted LDA model, and optionally a new column in the tcorpus (added by reference)

Usage

R6 method for class tCorpus. Use as tc$method (where tc is a tCorpus object).

```
lda_fit(feature, create_feature=NULL, K=50, num.iterations=500, alpha=50/K,
     eta=.01, burnin=250, context_level=c('document','sentence'), ...)
```

Examples

```
## Not run:
tc = create_tcorpus(sotu_texts, doc_column = 'id')
tc$preprocess('token', 'feature', remove_stopwords = TRUE, use_stemming = TRUE, min_freq=10)
set.seed(1)
m = tc$lda_fit('feature', create_feature = 'lda', K = 5, alpha = 0.1)

m
topicmodels::terms(m, 10)
tc$get()

## End(Not run)
```

---

tCorpus$preprocess *Preprocess feature*

Description

Preprocess feature

Arguments

column	the column containing the feature to be used as the input
new_column	the column to save the preprocessed feature. Can be a new column or overwrite an existing one.
lowercase	make feature lowercase
ngrams	create ngrams. The ngrams match the rows in the token data, with the feature in the row being the last token of the ngram. For example, given the features "this is an example", the third feature ("an") will have the trigram "this_is_an". Ngrams at the beginning of a context will have empty spaces. Thus, in the previous example, the second feature ("is") will have the trigram "_is_an".
ngram_context	Ngrams will not be created across contexts, which can be documents or sentences. For example, if the context_level is sentences, then the last token of sentence 1 will not form an ngram with the first token of sentence 2.

as_ascii	convert characters to ascii. This is particularly usefull for dealing with special characters.
remove_punctuation	remove (i.e. make NA) any features that are *only* punctuation (e.g., dots, comma's)
remove_stopwords	remove (i.e. make NA) stopwords. (!) Make sure to set the language argument correctly.
use_stemming	reduce features (tokens) to their stem
language	The language used for stopwords and stemming
min_freq	an integer, specifying minimum token frequency.
min_docfreq	an integer, specifying minimum document frequency.

Usage

R6 method for class tCorpus. Use as tc$method (where tc is a tCorpus object).

```
preprocess(column, new_column = column,
           lowercase = T, ngrams = 1, ngram_context=c('document', 'sentence'),
         as_ascii = F, remove_punctuation = T, remove_stopwords = F, use_stemming = F,
           language = 'english')
```

Examples

```
tc = create_tcorpus('I am a SHORT example sentence! That I am!')

## default is lowercase without punctuation
tc$preprocess('token', 'preprocessed_1')

## delete stopwords and perform stemming
tc$preprocess('token', 'preprocessed_2', remove_stopwords = TRUE, use_stemming = TRUE)

## filter on minimum frequency
tc$preprocess('token', 'preprocessed_3', min_freq=2)

## make ngrams
tc$preprocess('token', 'preprocessed_4', ngrams = 3)

tc$get()
```

---

tCorpus$read_text *Print tokens as text*

Description

Print tokens as text

Arguments

doc_id	The doc_ids of the documents to be printed.
column	The name of the column from which the text is printed.
meta_columns	The meta data that is printed at the top of each text.

Usage

R6 method for class tCorpus. Use as tc$method (where tc is a tCorpus object).

```
read_text(doc_id, column='token', meta_columns = self$meta_names)
```

Examples

```
d = data.frame(text = c('First text', 'Second text', 'Third text'),
medium = c('A','A','B'),
date = c('2010-01-01','2010-02-01','2010-03-01'))
tc = create_tcorpus(d)

tc$read_text(1)
tc$read_text(2)
tc$read_text(1:3)
```

---

tCorpus$search_contexts

Search for documents or sentences using Boolean queries

Description

Search for documents or sentences using Boolean queries

Arguments

query	A character string that is a query. See details for available query operators and modifiers. Can be multiple queries (as a vector), in which case it is recommended to also specifiy the code argument, to label results.
code	If given, used as a label for the results of the query. Especially usefull if multiple queries are used.
feature	The name of the feature column
context_level	Select whether the queries should occur within while "documents" or specific "sentences". Returns results at the specified level.
verbose	If TRUE, progress messages will be printed

Details

Brief summary of the query language

The following operators and modifiers are supported:

- The standaard Boolean operators: AND, OR and NOT. As a shorthand, an empty space can be used as an OR statement, so that "this that those" means "this OR that OR those". NOT statements stricly mean AND NOT, so should only be used between terms. If you want to find *everything except* certain terms, you can use * (wildcard for *anything*) like this: "* NOT (this that those)".
- For complex queries parentheses can (and should) be used. e.g. '(spam AND eggs) NOT (fish and (chips OR albatros))
- Wildcards ? and *. The questionmark can be used to match 1 unknown character or no character at all, e.g. "?at" would find "cat", "hat" and "at". The asterisk can be used to match any number of unknown characters. Both the asterisk and questionmark can be used at the start, end and within a term.
- Multitoken strings, or exact strings, can be specified using quotes. e.g. "united states"
- tokens within a given token distance can be found using quotes plus tilde and a number specifiying the token distance. e.g. "climate chang*"~10
- Alternatively, angle brackets (<>) can be used instead of quotes, which also enables nesting exact strings in proximity/window search
- Queries are not case sensitive, but can be made so by adding the ~s flag. e.g. COP~s only finds "COP" in uppercase. The ~s flag can also be used on quotes to make all terms within quotes case sensitive, and this can be combined with the token proximity flag. e.g. "Marco Polo"~s10

Usage

R6 method for class tCorpus. Use as tc$method (where tc is a tCorpus object).

```
search_contexts(query, code = NULL, feature = 'token', context_level = c('document','sentence'), verb
```

Examples

```
text = c('A B C', 'D E F. G H I', 'A D', 'GGG')
tc = create_tcorpus(text, doc_id = c('a','b','c','d'), split_sentences = TRUE)
tc$get() ## (example uses letters instead of words for simple query examples)

hits = tc$search_contexts(c('query label# A AND B', 'second query# (A AND Q) OR ("D E") OR I'))
hits          ## print shows number of hits
hits$hits     ## hits is a list, with hits$hits being a data.frame with specific contexts
summary(hits) ## summary gives hits per query

## sentence level
hits = tc$search_contexts(c('query label# A AND B', 'second query# (A AND Q) OR ("D E") OR I'),
                          context_level = 'sentence')
hits$hits     ## hits is a list, with hits$hits being a data.frame with specific contexts
```

```
## query language examples

## single term
tc$search_contexts('A')$hits

tc$search_contexts('G*')$hits    ## wildcard *
tc$search_contexts('*G')$hits    ## wildcard *
tc$search_contexts('G*G')$hits   ## wildcard *

tc$search_contexts('G?G')$hits   ## wildcard ?
tc$search_contexts('G?')$hits    ## wildcard ? (no hits)

## boolean
tc$search_contexts('A AND B')$hits
tc$search_contexts('A AND D')$hits
tc$search_contexts('A AND (B OR D)')$hits

tc$search_contexts('A NOT B')$hits
tc$search_contexts('A NOT (B OR D)')$hits


## sequence search (adjacent words)
tc$search_contexts('"A B"')$hits
tc$search_contexts('"A C"')$hits ## no hit, because not adjacent

tc$search_contexts('"A (B OR D)"')$hits ## can contain nested OR
## cannot contain nested AND or NOT!!

tc$search_contexts('<A B>')$hits ## can also use <> instead of "".

## proximity search (using ~ flag)
tc$search_contexts('"A C"~5')$hits ## A AND C within a 5 word window
tc$search_contexts('"A C"~1')$hits ## no hit, because A and C more than 1 word apart

tc$search_contexts('"A (B OR D)"~5')$hits ## can contain nested OR
tc$search_contexts('"A <B C>"~5')$hits    ## can contain nested sequence (must use <>)
tc$search_contexts('<A <B C>>~5')$hits   ## (<> is always OK, but cannot nest quotes in quotes)
## cannot contain nested AND or NOT!!


## case sensitive search
tc$search_contexts('g')$hits     ## normally case insensitive
tc$search_contexts('g~s')$hits   ## use ~s flag to make term case sensitive

tc$search_contexts('(a OR g)~s')$hits   ## use ~s flag on everything between parentheses
tc$search_contexts('(a OR G)~s')$hits   ## use ~s flag on everything between parentheses

tc$search_contexts('"a b"~s')$hits   ## use ~s flag on everything between quotes
tc$search_contexts('"A B"~s')$hits   ## use ~s flag on everything between quotes
```

`tCorpus$search_features`

Find tokens using a Lucene-like search query

Description

Search tokens in a tokenlist using Lucene-like queries. For a detailed explanation of the query language, see the details below.

Arguments

`query`	A character string that is a query. See details for available query operators and modifiers. Can be multiple queries (as a vector), in which case it is recommended to also specifiy the code argument, to label results.
`code`	The code given to the tokens that match the query (usefull when looking for multiple queries). Can also put code label in query with # (see details)
`feature`	The name of the feature column within which to search.
`mode`	There are two modes: "unique_hits" and "features". The "unique_hits" mode prioritizes finding full and unique matches., which is recommended for counting how often a query occurs. However, this also means that some tokens for which the query is satisfied might not assigned a hit_id. The "features" mode, instead, prioritizes finding all tokens, which is recommended for coding coding features (the code_features and search_recode methods always use features mode).
`context_level`	Select whether the queries should occur within while "documents" or specific "sentences".
`keep_longest`	If TRUE, then overlapping in case of overlapping queries strings in unique_hits mode, the query with the most separate terms is kept. For example, in the text "mr. Bob Smith", the query [smith OR "bob smith"] would match "Bob" and "Smith". If keep_longest is FALSE, the match that is used is determined by the order in the query itself. The same query would then match only "Smith".
`as_ascii`	if TRUE, perform search in ascii.
`verbose`	If TRUE, progress messages will be printed

Details

Brief summary of the query language

The following operators and modifiers are supported:

- The standaard Boolean operators: AND, OR and NOT. As a shorthand, an empty space can be used as an OR statement, so that "this that those" means "this OR that OR those". NOT statements stricly mean AND NOT, so should only be used between terms. If you want to find *everything except* certain terms, you can use * (wildcard for *anything*) like this: "* NOT (this that those)".
- For complex queries parentheses can (and should) be used. e.g. '(spam AND eggs) NOT (fish and (chips OR albatros))

- Wildcards ? and *. The questionmark can be used to match 1 unknown character or no character at all, e.g. "?at" would find "cat", "hat" and "at". The asterisk can be used to match any number of unknown characters. Both the asterisk and questionmark can be used at the start, end and within a term.
- Multitoken strings, or exact strings, can be specified using quotes. e.g. "united states"
- tokens within a given token distance can be found using quotes plus tilde and a number specifiying the token distance. e.g. "climate chang*"~10
- Alternatively, angle brackets (<>) can be used instead of quotes, which also enables nesting exact strings in proximity/window search
- Queries are not case sensitive, but can be made so by adding the ~s flag. e.g. COP~s only finds "COP" in uppercase. The ~s flag can also be used on parentheses or quotes to make all terms within case sensitive, and this can be combined with the token proximity flag. e.g. "Marco Polo"~s10
- The ~g (ghost) flag can be used to mark a term (or all terms within parentheses/quotes) as a ghost term. This has two effects. Firstly, features that match the query term will not be in the results. This is usefull if a certain term is important for getting reliable search results, but not conceptually relevant. Secondly, ghost terms can be used multiple times, in different query hits (only relevant in unique_hits mode). For example, in the text "A B C", the query 'A~g AND (B C)' will return both B and C as separate hit, whereas 'A AND (B C)' will return A and B as a single hit.
- A code label can be included at the beginning of a query, followed by a # to start the query (label# query). Note that to search for a hashtag symbol, you need to escape it with \ (double \ in R character vector)
- Aside from the feature column (specified with the feature argument) a query can include any column in the token data. To manually select a column, use 'columnname: ' at the start of a query or nested query (i.e. between parentheses or quotes). See examples for clarification.

Usage

R6 method for class tCorpus. Use as tc$method (where tc is a tCorpus object).

```
search_features(query, code = NA, feature = 'token',
                mode = c('unique_hits','features'), verbose = F)
```

Examples

```
text = c('A B C', 'D E F. G H I', 'A D', 'GGG')
tc = create_tcorpus(text, doc_id = c('a','b','c','d'), split_sentences = TRUE)
tc$get() ## (example uses letters instead of words for simple query examples)

hits = tc$search_features(c('query label# A AND B', 'second query# (A AND Q) OR ("D E") OR I'))
hits          ## print shows number of hits
hits$hits     ## hits is a list, with hits$hits being a data.frame with specific features
summary(hits) ## summary gives hits per query

## sentence level
hits = tc$search_features(c('query label# A AND B', 'second query# (A AND Q) OR ("D E") OR I'),
```

```
                          context_level = 'sentence')
hits$hits     ## hits is a list, with hits$hits being a data.frame with specific features


## query language examples

## single term
tc$search_features('A')$hits

tc$search_features('G*')$hits    ## wildcard *
tc$search_features('*G')$hits    ## wildcard *
tc$search_features('G*G')$hits   ## wildcard *

tc$search_features('G?G')$hits   ## wildcard ?
tc$search_features('G?')$hits    ## wildcard ? (no hits)

## boolean
tc$search_features('A AND B')$hits
tc$search_features('A AND D')$hits
tc$search_features('A AND (B OR D)')$hits

tc$search_features('A NOT B')$hits
tc$search_features('A NOT (B OR D)')$hits


## sequence search (adjacent words)
tc$search_features('"A B"')$hits
tc$search_features('"A C"')$hits ## no hit, because not adjacent

tc$search_features('"A (B OR D)"')$hits ## can contain nested OR
## cannot contain nested AND or NOT!!

tc$search_features('<A B>')$hits ## can also use <> instead of "".

## proximity search (using ~ flag)
tc$search_features('"A C"~5')$hits ## A AND C within a 5 word window
tc$search_features('"A C"~1')$hits ## no hit, because A and C more than 1 word apart

tc$search_features('"A (B OR D)"~5')$hits ## can contain nested OR
tc$search_features('"A <B C>"~5')$hits    ## can contain nested sequence (must use <>)
tc$search_features('<A <B C>>~5')$hits    ## <> is always OK, but cannot nest "" in ""
## cannot contain nested AND or NOT!!

## case sensitive search (~s flag)
tc$search_features('g')$hits     ## normally case insensitive
tc$search_features('g~s')$hits   ## use ~s flag to make term case sensitive

tc$search_features('(a OR g)~s')$hits   ## use ~s flag on everything between parentheses
tc$search_features('(a OR G)~s')$hits

tc$search_features('"a b"~s')$hits   ## use ~s flag on everything between quotes
tc$search_features('"A B"~s')$hits   ## use ~s flag on everything between quotes
```

```
## ghost terms (~g flag)
tc$search_features('A AND B~g')$hits   ## ghost term (~g) has to occur, but is not returned
tc$search_features('A AND Q~g')$hits    ## no hi

# (can also be used on parentheses/quotes/anglebrackets for all nested terms)


## "unique_hits" versus "features" mode
tc = create_tcorpus('A A B')

tc$search_features('A AND B')$hits ## in "unique_hits" (default), only match full queries
# (B is not repeated to find a second match of A AND B)

tc$search_features('A AND B', mode = 'features')$hits ## in "features", match any match
# (note that hit_id in features mode is irrelevant)

# ghost terms (used for conditions) can be repeated
tc$search_features('A AND B~g')$hits

## Not run:
## advanced queries
tc = tokens_to_tcorpus(corenlp_tokens, doc_col = 'doc_id',
                       sentence_col = 'sentence', token_id_col = 'id')
head(tc$get()) ## search in multiple feature columns with "columnname: "

## using the sub/flag query to find only mary as a direct object
hits = tc$search_features('mary~{relation: dobj}', context_level = 'sentence')
hits$hits

## add a second sub query
hits = tc$search_features('mary~{relation: dobj, parent: 12 20}', context_level = 'sentence')
hits$hits

## selecting from a different column without changing the feature column
## (can be used to combine columns)
hits = tc$search_features('relation: nsubj')
hits$hits

hits = tc$search_features('(relation: nsubj) AND mary~g{relation: dobj}',
                          context_level = 'sentence')
hits$hits

## sequence: nsubj say*
hits = tc$search_features('"(relation: nsubj) say*"')
hits$hits

## End(Not run)
```

---

tCorpus$search_recode *Recode features in a tCorpus based on a search string*

Description

Search features (see tCorpus$search_features) and replace features with a new value

Arguments

feature	The feature in which to search
new_value	the character string with which all features that are found are replaced
query	See tCorpus$search_features for the query parameters
...	Additional search_features parameters. See tCorpus$search_features

Usage

R6 method for class tCorpus. Use as tc$method (where tc is a tCorpus object).

```
search_recode(feature, new_value, keyword, condition = NA, condition_once = F)
```

---

tCorpus$semnet *Create a semantic network based on the co-occurence of tokens in documents*

Description

This function calculates the co-occurence of features and returns a network/graph in the igraph format, where nodes are tokens and edges represent the similarity/adjacency of tokens. Co-occurence is calcuated based on how often two tokens occured within the same document (e.g., news article, chapter, paragraph, sentence). The semnet_window() function can be used to calculate co-occurrence of tokens within a given token distance.

Arguments

feature	The name of the feature column
measure	The similarity measure. Currently supports: "con_prob" (conditional probability), "con_prob_weighted", "cosine" similarity, "count_directed" (i.e number of cooccurrences) and "count_undirected" (same as count_directed, but returned as an undirected network, chi2 (chi-square score))
context_level	Determine whether features need to co-occurr within "documents" or "sentences"
backbone	If True, add an edge attribute for the backbone alpha
n.batches	If a number, perform the calculation in batches

Usage

R6 method for class tCorpus. Use as tc$method (where tc is a tCorpus object).

```
semnet(feature, measure = c('con_prob', 'con_prob_weighted', 'cosine', 'count_directed', 'count_undir
       context_level = c('document','sentence'), backbone=F, n.batches=NA)
```

Examples

```
text = c('A B C', 'D E F. G H I', 'A D', 'GGG')
tc = create_tcorpus(text, doc_id = c('a','b','c','d'), split_sentences = TRUE)

g = tc$semnet('token')
g
igraph::get.data.frame(g)
## Not run: plot_semnet(g)
```

---

tCorpus$semnet_window *Create a semantic network based on the co-occurence of tokens in token windows*

Description

This function calculates the co-occurence of features and returns a network/graph in the igraph format, where nodes are tokens and edges represent the similarity/adjacency of tokens. Co-occurence is calcuated based on how often two tokens co-occurr within a given token distance.

Arguments

feature	The name of the feature column
measure	The similarity measure. Currently supports: "con_prob" (conditional probability), "cosine" similarity, "count_directed" (i.e number of cooccurrences) and "count_undirected" (same as count_directed, but returned as an undirected network, chi2 (chi-square score))
context_level	Determine whether features need to co-occurr within "documents" or "sentences"
window.size	The token distance within which features are considered to co-occurr
direction	Determine whether co-occurrence is assymmetricsl ("<>") or takes the order of tokens into account. If direction is '<', then the from/x feature needs to occur before the to/y feature. If direction is '>', then after.
backbone	If True, add an edge attribute for the backbone alpha
n.batches	If a number, perform the calculation in batches
set_matrix_mode	Advanced feature. There are two approaches for calculating window co-occurrence. One is to measure how often a feature occurs within a given token window, which can be calculating by calculating the inner product of a matrix that contains the exact position of features and a matrix that contains the occurrence window. We refer to this as the "positionXwindow" mode. Alternatively, we can measure how much the windows of features overlap, for which take the inner product of two window matrices. By default, semnet_window takes the mode that we deem most appropriate for the similarity measure. Substantially, the positionXwindow approach has the advantage of being very easy to interpret (e.g. how likely is feature "Y" to occurr within 10 tokens from feature "X"?). The windowXwindow mode, on the other hand, has the interesting feature that

similarity is stronger if tokens co-occurr more closely together (since then their windows overlap more). Currently, we only use the windowXwindow mode for cosine similarity. By using the set_matrix_mode parameter you can override this.

Usage

R6 method for class tCorpus. Use as tc$method (where tc is a tCorpus object).

```
semnet_window(feature, measure = c('con_prob', 'cosine', 'count_directed', 'count_undirected', 'chi2'
           context_level = c('document','sentence'), window.size = 10, direction = '<>',
           backbone = F, n.batches = 5, set_matrix_mode = c(NA, 'windowXwindow', 'positionXwindow'))
```

Examples

```
text = c('A B C', 'D E F. G H I', 'A D', 'GGG')
tc = create_tcorpus(text, doc_id = c('a','b','c','d'), split_sentences = TRUE)

g = tc$semnet_window('token', window.size = 1)
g
igraph::get.data.frame(g)
## Not run: plot_semnet(g)
```

---

tCorpus$set *Modify the token and meta data.tables of a tCorpus*

Description

Modify the token/meta data.table by setting the values of one (existing or new) column. The subset argument can be used to modify only subsets of columns, and can be a logical vector (select TRUE rows), numeric vector (indices of TRUE rows) or logical expression (e.g. pos == 'noun'). If a new column is made whie using a subset, then the rows outside of the selection are set to NA.

Arguments

column	Name of a new column (to create) or existing column (to transform)
value	An expression to be evaluated within the token/meta data, or a vector of the same length as the number of rows in the data. Note that if a subset is used, the length of value should be the same as the length of the subset (the TRUE cases of the subset expression) or a single value.
subset	logical expression indicating rows to keep in the tokens data or meta data

Usage

R6 method for class tCorpus. Use as tc$method (where tc is a tCorpus object).

```
set(column, value, subset)

set_meta(column, value, subset)
```

Examples

```
tc = create_tcorpus(sotu_texts, doc_column = 'id')

tc$get()  ## show original

## create new column
i <- 1:tc$n
tc$set(column = 'i', i)
## create new column based on existing column(s)
tc$set(column = 'token_upper', toupper(token))
## use subset to modify existing column
tc$set('token', paste0('***', token, '***'), subset = token_id == 1)
## use subset to create new column with NA's
tc$set('second_token', token, subset = token_id == 2)

tc$get()  ## show after set


##### use set for meta data with set_meta
tc$set_meta('party_pres', paste(party, president, sep=': '))
tc$get_meta()
```

---

tCorpus$set_levels *Change levels of factor columns*

Description

For factor columns, the levels can be changed directly (and by reference). This is particularly usefull for fast preprocessing (e.g., making tokens lowercase,)

Arguments

column	the name of the column
levels	The new levels

Usage

R6 method for class tCorpus. Use as tc$method (where tc is a tCorpus object).

```
set_levels(column, levels)


set_meta_levels(column, levels)
```

Examples

```
tc = create_tcorpus(c('Text one first sentence. Text one second sentence', 'Text two'))

## change factor levels of a column in the token data
unique_tokens <- tc$get_levels('token')
tc$set_levels('token', toupper(unique_tokens))
tc$get()
```

---

tCorpus$set_name *Change column names of data and meta data*

Description

Change column names of data and meta data

Arguments

oldname	the current/old column name
newname	the new column name

Usage

R6 method for class tCorpus. Use as tc$method (where tc is a tCorpus object).

```
set_name(oldname, newname)

set_meta_name(oldname, newname)
```

Examples

```
tc = create_tcorpus(sotu_texts, doc_column = 'id')

## change column name in token data
tc$names ## original column names
tc$set_name(oldname = 'token', newname = 'word')
tc$get()

## change column name in meta data
tc$meta_names ## original column names
tc$set_meta_name(oldname = 'party', newname = 'clan')
tc$set_meta_name(oldname = 'president', newname = 'clan leader')
tc$get_meta()
```

`tCorpus$subset` *Subset a tCorpus*

Description

Returns the subset of a tCorpus. The selection can be made separately (and simultaneously) for the token data (using subset) and the meta data (using subset_meta). The subset arguments work according to the subset.data.table function.

Important!! Note that subset is performed by reference. In other words, when performed, subset will delete the rows from the tCorpus, instead of returning a new tCorpus (see example for clarification). This is the standard behaviour, because it is much more efficient. If you want to create a subset of a copy of the tCorpus, you can set the copy argument to TRUE.

Subset can also be used to select rows based on token/feature frequences. This is a common step in corpus analysis, where it often makes sense to ignore very rare and/or very frequent tokens. To do so, there are several special functions that can be used within a subset call. The freq_filter() and docfreq_filter() can be used to filter terms based on term frequency and document frequency, respectively. (see examples)

The subset_meta() method is an alternative for using subset(subset_meta = ...), that is added for consistency with the other _meta accessor methods.

Note that you can also use the tCorpus$feature_subset method if you want to filter out low/high frequency tokens, but do not want to delete the rows in the tCorpus.

Arguments

`subset`	logical expression indicating rows to keep in the tokens data.
`subset_meta`	logical expression indicating rows to keep in the document meta data.
`window`	If not NULL, an integer specifiying the window to be used to return the subset. For instance, if the subset contains token 10 in a document and window is 5, the subset will contain token 5 to 15. Naturally, this does not apply to subset_meta.
`copy`	If TRUE, the method returns a new tCorpus object instead of subsetting the current one. This is added for convenience when analyzing a subset of the data. e.g., tc_nyt = tc$subset_meta(medium == "New_York_Times", copy=T)

Usage

R6 method for class tCorpus. Use as tc$method (where tc is a tCorpus object).

```
subset(subset = NULL, subset_meta = NULL,
       window = NULL, copy = F)
subset_meta(subset = NULL, copy = F)
```

Examples

```
tc = create_tcorpus(sotu_texts, doc_column = 'id')
tc$n ## original number of tokens

## select only first 20 tokens per document
tc$subset(token_id < 20)

tc$n ## number of tokens after subset

## note that the return value is not assigned to tc, or to a new name.
## rather, tc is changed by reference. To subset a copy of tc (the more classic R way),
## the copy argument can be used. The following line creates tc2 as a copy of tc,
## with only the first 10 tokens per document
tc2 <- tc$subset(token_id < 10, copy=TRUE)

tc$n   ## unchanged
tc2$n  ## subset of tc

## you can filter on term frequency and document frequency with the freq_filter() and
## docfreq_filter() functions
tc = create_tcorpus(sotu_texts, doc_column = 'id')
tc$subset( freq_filter(token, min = 20, max = 100) )
tc$get()


###### subset can be used for meta data by using the subset_meta argument, or the subset_meta method
tc$n_meta
tc$subset(subset_meta = president == 'Barack Obama')
tc$n_meta
tc$subset_meta(date == '2013-02-12')
tc$n_meta
```

---

tCorpus$subset_query *Subset tCorpus token data using a query*

Description

A convenience function that searches for contexts (documents, sentences), and uses the results to subset the tCorpus token data.

See the documentation for subset for an explanation of the query language.

Arguments

query	A character string that is a query. See tCorpus$search_contexts for query syntax.
feature	The name of the feature columns on which the query is used.
context_level	Select whether the query and subset are performed at the document, sentence or (token)window level. If window is selected, the window size is specified in the window argument
window	if context_level is window, specifies the size of the window

Usage

R6 method for class tCorpus. Use as tc$method (where tc is a tCorpus object).

```
subset_query(query, feature = 'token', context_level = c('document','sentence'))
```

Examples

```
text = c('A B C', 'D E F. G H I', 'A D', 'GGG')
tc = create_tcorpus(text, doc_id = c('a','b','c','d'), split_sentences = TRUE)

## subset by reference
tc$subset_query('A')
tc$get_meta()

## using copy mechanic
tc2 = tc$subset_query('A AND D', copy=TRUE)

tc2$get_meta()

tc$get_meta() ## (unchanged)
```

---

tCorpus$top_features *Show top features*

Description

Show top features

Arguments

feature	The name of the feature
n	Return the top n features
group_by	A column in the token data to group the top features by. For example, if token data contains part-of-speech tags (pos), then grouping by pos will show the top n feature per part-of-speech tag.
group_by_meta	A column in the meta data to group the top features by.
return_long	if True, results will be returned in a long format. Default is a table, but this can be inconvenient if there are many grouping variables.

Usage

R6 method for class tCorpus. Use as tc$method (where tc is a tCorpus object).

```
top_features(feature, n = 10, group_by = NULL, group_by_meta = NULL, return_long = F
```

Examples

```
tc = tokens_to_tcorpus(corenlp_tokens, token_id_col = 'id')

tc$top_features('lemma')
tc$top_features('lemma', group_by = 'relation')
```

---

tCorpus_compare *Corpus comparison*

Description

(back to overview)

Details

Compare vocabulary of two corpora

$compare_corpus()	Compare vocabulary of one tCorpus to another
$compare_subset()	Compare subset of a tCorpus to the rest of the tCorpus

---

tCorpus_create *Creating a tCorpus*

Description

(back to overview)

Details

Create a tCorpus

create_tcorpus()	Create a tCorpus from raw text input
tokens_to_tcorpus()	Create a tCorpus from a data.frame of already tokenized texts

---

tCorpus_data *Methods for viewing, modifying and subsetting tCorpus data*

Description

(back to overview)

Details

Get data

$get()	Get token data, with the possibility to select columns and subset
$get_meta()	Get meta data, with the possibility to select columns and subset
$dtm()	Create a document term matrix
$context()	Get a context vector. Currently supports documents or globally unique sentences.

Modify

The token and meta data can be modified with the set* and delete* methods. All modifications are performed by reference.

$set()	Modify the token data by setting the values of one (existing or new) column.
$set_meta()	The set method for the document meta data
$set_levels()	Change the levels of factor columns.
$set_meta_levels()	Change the levels of factor columns in the meta data
$set_name()	Modify column names of token data.
$set_meta_name()	Delete columns in the meta data
$delete_columns()	Delete columns.
$delete_meta_columns()	Delete columns in the meta data

Modifying is restricted in certain ways to ensure that the data always meets the assumptions required for tCorpus methods. tCorpus automatically tests whether assumptions are violated, so you don't have to think about this yourself. The most important limitations are that you cannot subset or append the data. For subsetting, you can use the tCorpus$subset method, and to add data to a tcorpus you can use the merge_tcorpora function.

Subsetting, merging/adding

$subset()	Modify the token and/or meta data using the tCorpus$subset function. A subset expression can be specified
$subset_meta()	For consistency with other *_meta methods
$subset_query()	Subset the tCorpus based on a query, as used in $search_contexts

Fields

For the sake of convenience, the number of rows and column names of the data and meta data.tables can be accessed directly. This is also faster and more memory efficient than using nrows() and colnames() on the data and meta fields, because those have to copy the data.tables.

$n	The number of tokens (i.e. rows in the data)
$n_meta	The number of documents (i.e. rows in the document meta data)
$names	The names of the token data columns
$names_meta	The names of the document meta data columns

`tCorpus_docsim` *Document similarity*

Description

(back to overview)

Details

Compare documents, and perform similarity based deduplication

$compare_documents()	Compare documents
$deduplicate()	Remove duplicate documents

`tCorpus_features` *Preprocessing, subsetting and analyzing features*

Description

(back to overview)

Details

Pre-process features

$preprocess()	Create or modify a feature by preprocessing an existing feature
$feature_subset()	Similar to using subset, but instead of deleting rows it only sets rows for a specified feature to NA.

Inspect features

$feature_stats()	Create a data.frame with feature statistics
$top_features()	Show top features, optionally grouped by a given factor

---

`tCorpus_modify_by_reference`
Modify tCorpus by reference

Description

(back to overview)

Details

If any tCorpus method is used that changes the corpus (e.g., set, subset), the change is made by reference. This is very convenient when working with a large corpus, because it means that the corpus does not have to be copied when changes are made, which is slow and memory inefficient.

To illustrate, for a tCorpus object named ‘tc‘, the subset method can be called like this:

tc$subset(doc_id %in% selection)

The 'tc' object itself is now modified, and does not have to be assigned to a name, as would be the more common R philosophy. Like this:

tc = tc$subset(doc_id %in% selection)

The results of both lines of code are the same. The assignment in the second approach is not necessary, but doesn't harm either because tc$subset returns the modified corpus invisibly (see ?invisible if that sounds spooky).

Be aware, however, that the following does not work!!

tc2 = tc$subset(doc_id %in% selection)

In this case, tc2 does contain the subsetted corpus, but tc itself will also be subsetted!!

We force this approach on you, because it is faster and more memory efficient, which becomes crucial for large corpora. If you do want to make a copy, it has to be done explicitly with the copy() method.

tc2 = tc$copy()

For methods where copying is often usefull, such as subset, there is also a copy parameter.

tc2 = tc$subset(doc_id %in% selection, copy=TRUE)

Now, tc will not be subsetted itself, but will subset a copy of itself and return it to be assigned to tc2.

Note that tc is also modified by reference if the subset method (or any other method that modified the corpus) is called within a function. No matter where and how you call the method, tc itself will be subsetted unless you explicitly copy it first or set copy to True.

`tCorpus_querying` *Use Boolean queries to analyze the tCorpus*

Description

(back to overview)

Details

Feature-level queries

$search_features())	Search for features based on keywords and conditions
$code_features())	Add a column to the token data based on feature search results
$search_recode()	Use the search_features query syntax to recode features
$feature_associations()	Given a query, get words that often co-occur nearby
$kwic()	Get keyword-in-context (kwic) strings

Context-level queries

$search_contexts()	Search for documents or sentences using Lucene-like queries
$subset_query()	use the search_contexts query syntax to subset the tCorpus

tCorpus_semnet *Feature co-occurrence based semantic network analysis*

Description

(back to overview)

Details

Create networks

$semnet)	Feature co-occurrence within contexts (documents, sentences)
$semnet_window()	Feature co-occurrence within a specified token distance

Support functions for analyzing and visualizing the semantic network

ego_semnet()	Create an ego network from an Igraph network
plot_semnet()	Convenience function for visualizing an Igraph network, specialized for semantic networks

tCorpus_topmod *Topic modeling*

Description

(back to overview)

Details

Train a topic model

$lda_fit()	Latent Dirichlet Allocation

tokens_to_tcorpus *Create a tcorpus based on tokens (i.e. preprocessed texts)*

Description

Create a tcorpus based on tokens (i.e. preprocessed texts)

Usage

```
tokens_to_tcorpus(tokens, doc_col = "doc_id", token_id_col = "token_id",
```

```
  sentence_col = "sentence", meta = NULL, meta_cols = NULL,
  feature_cols = NULL, sent_is_local = T, token_is_local = T)
```

Arguments

tokens	A data.frame in which rows represent tokens, and columns indicate (at least) the document in which the token occured (doc_col) and the position of the token in that document or globally (token_id_col)
doc_col	The name of the column that contains the document ids/names
token_id_col	The name of the column that contains the positions of tokens. If NULL, it is assumed that the data.frame is ordered by the order of tokens and does not contain gaps (e.g., filtered out tokens)
sentence_col	Optionally, the name of the column that indicates the sentences in which tokens occured.
meta	Optionally, a data.frame with document meta data. Needs to contain a column with the document ids (with the same name)
meta_cols	Alternatively, if there are document meta columns in the tokens data.table, meta_cols can be used to recognized them. Note that these values have to be unique within documents.
feature_cols	Optionally, specify which columns to include in the tcorpus. If NULL, all column are included (except the specified columns for documents, sentences and positions)
sent_is_local	Sentences in the tCorpus are assumed to be locally unique within documents. If sent_is_local is FALSE, then sentences are transformed to be locally unique. However, it is then assumed that the first sentence in a document is sentence 1, which might not be the case if tokens (input) is a subset.
token_is_local	Same as sent_is_local, but for token_id. Note that if a parent column is present, it will not be changed along.

Examples

```
head(corenlp_tokens)

tc = tokens_to_tcorpus(corenlp_tokens, doc_col = 'doc_id',
                       sentence_col = 'sentence', token_id_col = 'id')
tc

meta = data.frame(doc_id = 1, medium = 'A', date = '2010-01-01')
tc = tokens_to_tcorpus(corenlp_tokens, doc_col = 'doc_id',
                       sentence_col = 'sentence', token_id_col = 'id', meta=meta)
tc
```

`tokenWindowOccurence` *Gives the window in which a term occured in a matrix.*

Description

This function returns the occurence of tokens (position.matrix) and the window of occurence (window.matrix). This format enables the co-occurence of tokens within sliding windows (i.e. token distance) to be calculated by multiplying position.matrix with window.matrix.

Usage

```
tokenWindowOccurence(tc, feature, context_level = c("document", "sentence"),
  window.size = 10, direction = "<>", distance_as_value = F,
  batch_rows = NULL, drop_empty_terms = T)
```

Arguments

`tc`	a tCorpus object
`feature`	The name of the feature column
`context_level`	Select whether to use "document" or "sentence" as context boundaries
`window.size`	The distance within which tokens should occur from each other to be counted as a co-occurence.
`direction`	a string indicating whether only the left ('<') or right ('>') side of the window, or both ('<>'), should be used.
`distance_as_value`	If True, the values of the matrix will represent the shorts distance to the occurence of a feature
`batch_rows`	Used in functions that call this function in batches
`drop_empty_terms`	If TRUE, emtpy terms (with zero occurence) will be dropped

Value

A list with two matrices. position.mat gives the specific position of a term, and window.mat gives the window in which each token occured. The rows represent the position of a term, and matches the input of this function (position, term and context). The columns represents terms.

Index