Package ‘corpustools’

October 12, 2022

Version 0.4.10
Date 2022-05-03
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 (>= 3.5.0)
Imports methods, wordcloud (>= 2.5), stringi, Rcpp (>= 0.12.12), R6, uudpipe (>= 0.8.3), digest, data.table (>= 1.10.4), quanteda (>= 1.5.1), igraph, tokenbrowser (>= 0.1.5), RNewsflow (>= 1.2.1), Matrix (>= 1.2), parallel, pbapply (>= 1.4), rsyntax (>= 0.1.1)
Suggests testthat, tm (>= 0.6), topicmodels, knitr, rmarkdown
LinkingTo Rcpp, RcppProgress
LazyData true
Encoding UTF-8
SystemRequirements C++11
License GPL-3
URL https://github.com/kasperwelbers/corpustools
RoxygenNote 7.1.2
VignetteBuilder knitr
NeedsCompilation yes
Repository CRAN
Date/Publication 2022-05-11 10:00:05 UTC
R topics documented:

add_multitoken_label ........................................... 4
aggregate_rsyntax .................................................. 5
agg_label .......................................................... 6
agg_tcorpus ....................................................... 7
as.tcorpus ........................................................ 8
as.tcorpus.default .............................................. 9
as.tcorpus.tCorpus .............................................. 9
backbone_filter ................................................... 10
browse_hits ........................................................ 11
browse_texts ...................................................... 12
calc_chi2 .......................................................... 14
compare_corpus .................................................... 15
compare_documents ............................................... 16
compare_subset ................................................... 18
corenlp_tokens .................................................... 19
count_tcorpus ..................................................... 19
create_tcorpus .................................................... 20
docfreq_filter ..................................................... 24
dtm_compare ....................................................... 25
dtm_wordcloud ..................................................... 26
ego_semnet ......................................................... 27
export_span_annotations ......................................... 28
feature_associations ............................................. 29
feature_stats ...................................................... 31
fold_rsyntax ....................................................... 32
freq_filter ........................................................ 33
get_dtm ........................................................... 33
get_global_i ....................................................... 36
get_kwic ........................................................... 36
get_stopwords ..................................................... 38
laplace ............................................................. 39
melt_quanteda_dict ............................................... 39
merge_tcorpora ................................................... 40
plot.contextHits .................................................. 41
plot.featureAssociations ....................................... 42
plot.featureHits .................................................. 42
plot.vocabularyComparison .................................... 43
plot_semnet ....................................................... 44
plot_words ........................................................ 46
preprocess_tokens ............................................... 47
print.contextHits ................................................ 49
print.featureHits ................................................ 49
print.tCorpus ..................................................... 50
refresh_tcorpus .................................................. 50
require_package .................................................. 51
search_contexts ................................................... 52
R topics documented:

- search_dictionary ........................................ 54
- search_features ........................................... 56
- semnet .................................................... 60
- semnet_window ........................................... 61
- set_network_attributes ................................... 62
- sgt ......................................................... 63
- show_udpipe_models ...................................... 64
- sotu_texts ............................................... 64
- stopwords_list ............................................ 65
- subset.tCorpus ........................................... 65
- subset_query .............................................. 66
- summary.contextHits ..................................... 67
- summary.featureHits ..................................... 67
- summary.tCorpus .......................................... 68
- tCorpus .................................................... 68
- tCorpus$annotate_rsyntax ................................ 69
- tCorpus$code_dictionary ................................ 70
- tCorpus$code_features ................................... 72
- tCorpus$context ......................................... 73
- tCorpus$deduplicate .................................... 73
- tCorpus$delete_columns .................................. 75
- tCorpus$feats_to_columns ................................ 76
- tCorpus$feature_subset .................................. 76
- tCorpus$fold_rsyntax ..................................... 77
- tCorpus$get ............................................... 78
- tCorpus$lda_fit .......................................... 80
- tCorpus$merge ............................................ 81
- tCorpus$preprocess ...................................... 82
- tCorpus$replace_dictionary .............................. 83
- tCorpus$search_recode ..................................... 85
- tCorpus$set ............................................... 86
- tCorpus$set_levels ...................................... 87
- tCorpus$set_name ....................................... 88
- tCorpus$subset ........................................... 88
- tCorpus$subset_query .................................... 90
- tCorpus$subset_query .................................... 90
- tCorpus$udpipe_clauses .................................. 91
- tCorpus$udpipe_coref ................................... 92
- tCorpus$udpipe_quotes ................................... 92
- tCorpus_compare ......................................... 94
- tCorpus_create .......................................... 94
- tCorpus_data ............................................. 94
- tCorpus_docsim .......................................... 95
- tCorpus_features ......................................... 96
- tCorpus_modify_by_reference ............................. 96
- tCorpus_querying ...................................... 96
- tCorpus_semnet .......................................... 98
- tCorpus_topmod .......................................... 98
- tc_plot_tree ............................................. 98
add_multitoken_label

Choose and add multitoken strings based on multitoken categories

Description

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

Usage

```r
add_multitoken_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 multitoken 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 occurring string
Description

A method for aggregating rsyntax annotations. The intended purpose is to compute aggregate values for a given label in an annotation column.

For example, you used annotate_rsyntax to add a column with subject-predicate labels, and now you want to concatenate the tokens with these labels. With annotate_rsyntax you would first aggregate the subject tokens, then aggregate the predicate tokens. By default (txt = T) the column with concatenated tokens are added.

You can specify any aggregation function using any column in tc$tokens. So say you want to perform a sentiment analysis on the quotes of politicians. You first used annotate_rsyntax to create an annotation column ‘quote’, that has the labels ‘source’, ‘verb’, and ‘quote’. You also used code_dictionary to add a column with unique politician ID’s and a column with sentiment scores. Now you can aggregate the source tokens to get a single unique ID, and aggregate the quote tokens to get a single sentiment score.

Usage

aggregate_rsyntax(
    tc,
    annotation,
    ...,  
    by_col = NULL,
    txt = F,
    labels = NULL,
    rm_na = T
)

Arguments

tc a tCorpus
annotation The name of the rsyntax annotation column
... To aggregate columns for specific
by_col A character vector with other column names in tc$tokens to aggregate by.
txt If TRUE, add columns with concatenated tokens for each label. Can also be a character vector specifying for which specific labels to create this column
labels Instead of using all labels, a character vector of labels can be given
rm_na If TRUE, remove rows with only NA values

Value

A data.table
Examples

## Not run:
tc = tc_sotu_udpipe$copy()
tc$udpipe_clauses()

subject_verb_predicate = aggregate_rsyntax(tc, 'clause', txt=TRUE)
head(subject_verb_predicate)

## We can also add specific aggregation functions

## count number of tokens in predicate
aggregate_rsyntax(tc, 'clause',
    agg_label('predicate', n = length(token_id)))

## same, but with txt for only the subject label
aggregate_rsyntax(tc, 'clause', txt='subject',
    agg_label('predicate', n = length(token_id)))

## example application: sentiment scores for specific subjects

# first use queries to code subjects
tc$code_features(column = 'who',
    query = c('I# I~s <this president>',
    'we# we americans <american people>'))

# then use dictionary to get sentiment scores
dict = melt_quanteda_dict(quanteda::data_dictionary_LSD2015)
dict$sentiment = ifelse(dict$code %in% c('negative', 'neg_positive'), -1, 1)
tc$code_dictionary(dict)

sent = aggregate_rsyntax(tc, 'clause', txt='predicate',
    agg_label('subject', subject = na.omit(who)[1]),
    agg_label('predicate', sentiment = mean(sentiment, na.rm=TRUE)))

head(sent)
sent[,list(sentiment=mean(sentiment, na.rm=TRUE), n=.N), by='subject']

## End(Not run)

### agg_label

#### Helper function for aggregate_rsyntax

**Description**

This function is used within the `aggregate_rsyntax` function to facilitate aggregating by specific labels.

**Usage**

```r
agg_label(label, ...)
```
**agg_tcorpus**

### Arguments

- **label**
  - The rsyntax label. Needs to be an existing value in the annotation column (as specified when calling *aggregate_rsyntax*).

- **...**
  - Specify the new aggregated columns in name-value pairs. The name is the name of the new column, and the value should be a function over a column in $tokens. For example: `subject = paste(token, collapse = ' ')` would create the column 'subject', of which the values are the concatenated tokens. See examples for more.

### Value

Not relevant. Should only be used within *aggregate_rsyntax*.

### Examples

```r
tc = tc_sotu_udpipe$copy()
tc$udpipe_clauses()

## count number of tokens in predicate
aggregate_rsyntax(tc, 'clause', txt=FALSE,
  agg_label('predicate', n = length(token_id)))
```

---

**Agg_tcorpus**

*Aggregate the tokens data*

### Description

This is a wrapper for the data.table aggregate function, for easy aggregation of the tokens data grouped by columns in the tokens or meta data. The .id argument is an important addition, because token annotation often contain values that span multiple rows.

### Usage

```r
agg_tcorpus(tc, ..., by = NULL, .id = NULL, wide = T)
```

### Arguments

- **tc**
  - A tCorpus

- **...**
  - The name of the aggregated column and the function over an existing column are given as a name value pair. For example, `count = length(token)` will count the number of tokens in each group, and `sentiment = mean(sentiment, na.rm=T)` will calculate the mean score for a column with sentiment scores.

- **by**
  - A character vector with column names from the tokens and/or meta data.
.id If an id column is given, only rows for which this id is not NA are used, and only one row for each id is used. This prevents double counting of values in annotations that span multiple rows. For example, a sentiment dictionary can match the tokens "not good", in which case the sentiment score (-1) will be assigned to both tokens. These annotations should have an _id column that indicates the unique matches.

wide Should results be in wide or long format?

Value
A data table

Examples

tc = create_tcorpus(sotu_texts, doc_col='id')

library(quanteda)
dict = data_dictionary_LSD2015
dict = melt_quanteda_dict(dict)
dict$sentiment = ifelse(dict$code %in% c('positive','neg_negative'), 1, -1)
tc$code_dictionary(dict)

agg_tcorpus(tc, N = length(sentiment), sent = mean(sentiment), .id='code_id')
agg_tcorpus(tc, sent = mean(sentiment), .id='code_id', by='president')
agg_tcorpus(tc, sent = mean(sentiment), .id='code_id', by=c('president', 'token'))

---

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


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 clarification).
- **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_A01
- **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 arbitrary if there are many very weak ties, which is often the case in a co-occurrence 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-occurrence is never zero).
Value

A graph in the Igraph format

Examples

tc = create_tcorpus(sotu_texts, doc_column = 'id')
tc$preprocess('token', 'feature', remove_stopwords = TRUE, use_stemming = TRUE, min_docfreq = 10)

g = semnet_window(tc, '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)

browse_hits

View hits in a browser

Description

Creates a static HTML file to view the query hits in the tcorpus in full text mode.

Usage

browse_hits(
  tc,
  hits,
  token_col = "token",
  n = 500,
  select = c("first", "random"),
  header = "",
  subheader = NULL,
  meta_cols = NULL,
  seed = NA,
  view = T,
  filename = NULL
)

Arguments

tc a tCorpus
hits a featureHits object, as returned by search_features
token_col The name of the column in tc$tokens that contain the token text
If doc_ids is NULL, Only n of the results are printed (to prevent accidentally making huge browsers).

If n is smaller than the number of documents in tc, select determines how the n documents are selected

Optionally, a title presented at the top of the browser

Optionally, overwrite the subheader. By default the subheader reports the number of documents

A character vector with names of columns in tc$meta, used to only show the selected columns

If select is "random", seed can be used to set a random seed

If TRUE (default), view the browser in the Viewer window (turn off if this is not supported)

Optionally, save the browser at a specified location

The url for the file location is returned (invisibly)

```r
tc = create_tcorpus(sotu_texts, doc_column='id')
hits = search_features(tc, c("Terrorism# terroris*", "War# war*"))
browse_hits(tc, hits)
```

Create and view a full text browser

Creates a static HTML file to view the texts in the tcorpus in full text mode.

```r
browse_texts(
  tc, 
  doc_ids = NULL, 
  token_col = "token", 
  n = 500, 
  select = c("first", "random"), 
  header = "", 
  subheader = NULL, 
  highlight = NULL, 
  scale = NULL, 
  category = NULL,
)```
browse_texts

rsyntax = NULL,
value = NULL,
meta_cols = NULL,
seed = NA,
nav = NULL,
top_nav = NULL,
thres_nav = 1,
view = T,
highlight_col = "yellow",
scale_col = c("red", "blue", "green"),
filename = NULL
)

Arguments

tc a tCorpus
doc_ids A vector with document ids to view
token_col The name of the column in tc$tokens that contain the token text
n Only n of the results are printed (to prevent accidentally making huge browsers).
select If n is smaller than the number of documents in tc, select determines how the n
documents are selected
header Optionally, a title presented at the top of the browser
subheader Optionally, overwrite the subheader. By default the subheader reports the num-
ber of documents
highlight Highlight mode: provide the name of a numeric column in tc$tokens with values
between 0 and 1, used to highlight tokens. Can also be a character vector, in
which case all non-NA values are highlighted
scale Scale mode: provide the name of a numeric column in tc$tokens with values
between -1 and 1, used to color tokens on a scale (set colors with scale_col)
category Category mode: provide the name of a character or factor column in tc$tokens.
Each unique value will have its own color, and navigation for categories will be
added (nav cannot be used with this option)
rsyntax rsyntax mode: provide the name of an rsyntax annotation column (see annotate_rsyntax)
value rsyntax mode argument: if rsyntax mode is used, value can be a character vector
with values in the rsyntax annotation column. If used, only these values are fully
colored, and the other (non NA) values only have border colors.
meta_cols A character vector with names of columns in tc$meta, used to only show the
selected columns
seed If select is "random", seed can be used to set a random seed. After sampling the
seed is re-initialized with set.seed(NULL).
nav Optionally, a column in tc$meta to add navigation (only supports simple filtering
on unique values). This is not possible if category is used.
top_nav A number. If navigation based on token annotations is used, filters will only
apply to top x values with highest token occurrence in a document
calc_chi2

Vectorized computation of \( \chi^2 \) statistic for a 2x2 crosstab containing the values \( \begin{bmatrix} a & b \\ c & d \end{bmatrix} \)

**Description**

Vectorized computation of \( \chi^2 \) statistic for a 2x2 crosstab containing the values \( a, b \) \( c, d \)

**Usage**

\[
\text{calc_ch}i2(a, b, c, d, \text{correct} = \text{T}, \text{cochrans_criteria} = \text{F})
\]

**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
**compare_corpus**

*Compare tCorpus vocabulary to that of another (reference) tCorpus*

**Description**

Compare tCorpus vocabulary to that of another (reference) tCorpus

**Usage**

```r
compare_corpus(
  tc,
  tc_y,
  feature,
  smooth = 0.1,
  min_ratio = NULL,
  min_chi2 = NULL,
  is_subset = F,
  yates_cor = c("auto", "yes", "no"),
  what = c("freq", "docfreq", "cooccurrence")
)
```

**Arguments**

- `tc`: a `tCorpus`
- `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 probabilities. Here you can set the added (pseudocount) 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
- `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`
- `yates_cor`: mode for using yates correction 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
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 = compare_corpus(tc, bush, 'feature')
comp = comp[order(-comp$chi),]
head(comp)
plot(comp)

Description

Calculate the similarity of documents

Usage

compare_documents(
  tc,
  feature = "token",
  date_col = NULL,
  meta_cols = NULL,
  hour_window = c(24),
  measure = c("cosine", "overlap_pct"),
  min_similarity = 0,
  weight = c("norm_tfidf", "tfidf", "termfreq", "docfreq"),
  ngrams = NA,
  from_subset = NULL,
  to_subset = NULL,
  return_igraph = T,
  verbose = T
)

Arguments

  tc  A tCorpus
  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.
meta_cols  a character vector with columns in the meta data / docvars. If given, only documents for which these values are identical are compared

hour_window  A vector of length 1 or 2. If length is 1, the same value is used for the left and right side of the window. If length is 2, the first and second value determine the left and right side. For example, the value 12 will compare each document to all documents between the previous and next 12 hours, and c(-10, 36) will compare each document to all documents between the previous 10 and the next 36 hours.

measure  the similarity measure. Currently supports cosine similarity (symmetric) and overlap_pct (asymmetric)

min_similarity  A threshold for the similarity score

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

return_igraph  If TRUE, return as an igraph network. Otherwise, return as a list with the edge-list and meta data.

verbose  If TRUE, report progress

Value
An igraph graph in which nodes are documents and edges represent similarity scores

Examples

d = data.frame(text = c("a b c d e",
  'e f g h i j k",
  'a b c"),
  date = as.POSIXct(c('2010-01-01', '2010-01-01', '2012-01-01')))tc = create_tcorpus(d)
g = compare_documents(tc)
igraph::get.data.frame(g)

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

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

*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

**Usage**

```r
close_subset(
  tc,
  feature,
  subset_x = NULL,
  subset_meta_x = NULL,
  query_x = NULL,
  query_feature = "token",
  smooth = 0.1,
  min_ratio = NULL,
  min_chi2 = NULL,
  yates_cor = c("auto", "yes", "no"),
  what = c("freq", "docfreq", "cooccurrence")
)
```

**Arguments**

- **tc**  
a tCorpus

- **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 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 probabilities. Here you can set the added (pseudo-count) 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 correction 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

Examples

tc = create_tcorpus(sotu_texts, doc_column = 'id')
tc$preprocess('token', 'feature', remove_stopwords = TRUE, use_stemming = TRUE)
comp = compare_subset(tc, 'feature', subset_meta_x = president == 'Barack Obama')
comp = comp[order(-comp$chi),]
head(comp)
plot(comp)

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

corenlp_tokens          coreNLP example sentences

Description

coreNLP example sentences

Usage

data(corenlp_tokens)

Format

data.frame

count_tcorpus          Count results of search hits, or of a given feature in tokens

Description

Count results of search hits, or of a given feature in tokens
create_tcorpus

Usage

count_tcorpus(
  tc,
  meta_cols = NULL,
  hits = NULL,
  feature = NULL,
  count = c("documents", "tokens", "hits"),
  wide = T
)

Arguments

tc A tCorpus
meta_cols The columns in the meta data by which the results should be grouped
hits featureHits or contextHits (output of search_features, search_dictionary or search_contexts)
feature Instead of hits, a specific feature column can be selected.
count How should the results be counted? Number of documents, tokens, or unique hits. The difference between tokens and hits is that hits can encompass multiple tokens (e.g., "Bob Smith" is 1 hit and 2 tokens).
wide Should results be in wide or long format?

Value

A data table

Examples

tc = create_tcorpus(sotu_texts, doc_col="id")
hits = search_features(tc, c("US# <united states>", "Economy# econom*"))
count_tcorpus(tc, hits=hits)
count_tcorpus(tc, hits=hits, meta_cols='president')
count_tcorpus(tc, hits=hits, meta_cols='president', wide=FALSE)

create_tcorpus Create a tCorpus

Description

Create a tCorpus from raw text input. Input can be a character (or factor) vector, data.frame or quanteda corpus. If a data.frame is given, all columns other than the document id and text columns are included as meta data. If a quanteda corpus is given, the ids and texts are already specified, and the docvars will be included in the tCorpus as meta data.
create_tcorpus

Usage

create_tcorpus(x, ...)

## S3 method for class 'character'
create_tcorpus(
  x,
  doc_id = 1:length(x),
  meta = NULL,
  udpipe_model = NULL,
  split_sentences = F,
  max_sentences = NULL,
  max_tokens = NULL,
  udpipe_model_path = getwd(),
  udpipe_cache = 3,
  udpipe_cores = NULL,
  udpipe_batchsize = 50,
  use_parser = F,
  remember_spaces = F,
  verbose = T,
  ...
)

## S3 method for class 'data.frame'
create_tcorpus(
  x,
  text_columns = "text",
  doc_column = "doc_id",
  udpipe_model = NULL,
  split_sentences = F,
  max_sentences = NULL,
  max_tokens = NULL,
  udpipe_model_path = getwd(),
  udpipe_cache = 3,
  udpipe_cores = NULL,
  udpipe_batchsize = 50,
  use_parser = F,
  remember_spaces = F,
  verbose = T,
  ...
)

## S3 method for class 'factor'
create_tcorpus(x, ...)

## S3 method for class 'corpus'
create_tcorpus(x, ...)
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. If x (or a text_column in x) has leading or trailing whitespace, this is cut off (and you’ll get a warning about it).

- **...**: Arguments passed to create_tcorpus.character

- **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

- **udpipe_model**: Optionally, the name of a Universal Dependencies language model (e.g., "english-ewt", "dutch-alpino"), to use the udpipe package (udpipe_annotate) for natural language processing. You can use show_udpipe_models to get an overview of the available models. For more information about udpipe and performance benchmarks of the UD models, see the GitHub page of the udpipe package.

- **split_sentences**: Logical. If TRUE, the sentence number of tokens is also computed. (only if udpipe_model is not used)

- **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

- **udpipe_model_path**: If udpipe_model is used, this path will be used to look for the model, and if the model doesn’t yet exist it will be downloaded to this location. Defaults to working directory

- **udpipe_cache**: The number of persistent caches to keep for inputs of udpipe. The caches store tokens in batches. This way, if a lot of data has to be parsed, or if R crashes, udpipe can continue from the latest batch instead of start over. The caches are stored in the corpustools_data data folder (in udpipe_model_path). Only the most recent [udpipe_caches] caches will be stored.

- **udpipe_cores**: If udpipe_model is used, this sets the number of parallel cores. If not specified, will use the same number of cores as used by data.table (or limited to OMP_THREAD_LIMIT).

- **udpipe_batchsize**: In order to report progress and cache results, texts are parsed with udpipe in batches of 50. The price is that there will be some overhead for each batch, so for very large jobs it can be faster to increase the batchsize. If the number of texts divided by the number of parallel cores is lower than the batchsize, the texts are evenly distributed over cores.

- **use_parser**: If TRUE, use dependency parser (only if udpipe_model is used)

- **remember_spaces**: If TRUE, a column with spaces after each token and column with the start and end positions of tokens are included. Can turn it off for a bit more speed and less memory use, but some features won’t work.
create_tcorpus

verbose If TRUE, report progress. Only if x is large enough to require multiple sequential batches

text_columns if x is a data.frame, this specifies the column(s) that contains text. If multiple columns are used, they are pasted together separated by a double line break. If remember_spaces is true, a "field" column is also added that show the column name for each token, and the start/end positions are local within these fields

doc_column If x is a data.frame, this specifies the column with the document ids.

Details

By default, texts will only be tokenized, and basic preprocessing techniques (lowercasing, stemming) can be applied with the preprocess method. Alternatively, the udpipe package can be used to apply more advanced NLP preprocessing, by using the udpipe_model argument.

For certain advanced features you need to set remember_spaces to true. We are often used to forgetting all about spaces when we do bag-of-word type stuff, and that's sad. With remember_spaces, the exact position of each token is remembered, including what type of space follows the token (like a space or a line break), and what text field the token came from (if multiple text_columns are specified in create_tcorpus.data.frame)

Examples

```r
## ...
ntc = create_tcorpus(c('Text one first sentence. Text one second sentence', 'Text two'))
tc$tokens

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

## 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
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$tokens

## 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',
```
```r
split_sentences = TRUE)
tc
tc$tokens
## 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$tokens

library(quanteda)
create_tcorpus(data_corpus_inaugural)
```

---

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

```r
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.
dtm_compare

Examples

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

tc$tokens
tc$subset(subset = docfreq_filter(token, min=2))
tc$tokens

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    Laplace smoothing is used for the calculation of the probabilities. Here you can set the added (pseuocount) 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
  select_rows Alternative to using dtm.y. Has to be a vector with rownames, by which
  yates_cor  mode for using yates correction 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**

```r
dtm_wordcloud(
  dtm = NULL,
  nterms = 100,
  freq.fun = NULL,
  terms = NULL,
  freqs = NULL,
  scale = c(4, 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 frequencies (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 frequency to include (see wordcloud::wordcloud)
- `rot.per` the percentage of vertical words (see wordcloud::wordcloud)
- `...` other arguments passed to wordcloud::wordcloud
### Examples

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

dtm_wordcloud(dtm, nterms = 20)

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

---

**ego_semnet**  
*Create an ego network*

#### Description

Create an ego network from an igraph object.

#### Usage

```r
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 = semnet(tc, 'token')

igraph::get.data.frame(g)
plot_semnet(g)

## only keep nodes directly connected to given node
eg = ego_semnet(g, 'e')
igraph::get.data.frame(eg)
plot_semnet(eg)

## only keep edges directly connected to given node
eg = ego_semnet(g, 'e', only_filter_vertices = FALSE)
igraph::get.data.frame(eg)
plot_semnet(eg)

## only keep nodes connected to given node with a specified degree (i.e. distance)
eg = ego_semnet(g, 'e', depth = 2)
igraph::get.data.frame(eg)
plot_semnet(eg)
Usage

export_span_annotations(tc, variables)

Arguments

tc
A tCorpus, created with create_tcorpus, where remember_spaces must have been set to TRUE

variables
A character vector with variables (columns in tc$tokens) to export

Details

Note that if there are spans with gaps in them (e.g. based on proximity queries), they are split into different annotations. Thus some information can be lost.

Value

A data.table where each row is a span annotation, with columns: doc_id, variable, value, field, offset, length, text

Examples

tc = create_tcorpus(sotu_texts, c('president', 'text'), doc_column='id', remember_spaces=TRUE)
tc$code_features(c('war# war peace', 'us being# <(i we) (am are)>'))
export_span_annotations(tc, 'code')

---

feature_associations

Get common nearby features given a query or query hits

Description

Get common nearby features given a query or query hits

Usage

feature_associations(
  tc,
  feature,
  query = NULL,
  hits = NULL,
  query_feature = "token",
  window = 15,
  n = 25,
  min_freq = 1,
  sort_by = c("chi2", "ratio", "freq"),
  subset = NULL,
  subset_meta = NULL,
  include_self = F
)
feature_associations

Arguments

tc a tCorpus
feature The name of the feature column in $tokens
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 search_features can be
used.
query_feature If query is used, the column in $tokens on which the query is performed. By
default uses 'token'
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 organiza-
tion. 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 sub-
set_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 depen-
dent. 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).
include_self If True, include the feature itself in the output

Value

a data.frame

Examples

tc = create_tcorpus(sotu_texts, doc_column = 'id')
tc$preprocess()

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

## adjust window size
topf = feature_associations(tc, 'feature', '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 = search_features(tc, "'war terror'~10")
topf = feature_associations(tc, 'feature', hits = hits)
head(topf, 20) ## frequent words close to the combination of "war" and "terror" within 10 words

---

**feature_stats**  
*Feature statistics*

**Description**

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

**Usage**

```
feature_stats(tc, feature, context_level = c("document", "sentence"))
```

**Arguments**

- **tc**: a tCorpus
- **feature**: The name of the feature column
- **context_level**: Should results be returned at document or sentence level

**Value**

a data.frame

**Examples**

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

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

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

Fold rsyntax annotations

Description

If a tCorpus has rsyntax annotations (see annotate_rsyntax), it can be convenient to aggregate tokens that have a certain semantic label. For example, if you have a query for labeling "source" and "quote", you can add an aggregated value for the sources (such as a unique ID) as a column, and then remove the quote tokens.

Usage

fold_rsyntax(tc, annotation, by_label, ..., txt = F, rm_by = T)

Arguments

tc  A tCorpus
annotation  The name of an rsyntax annotation column
by_label  The labels in this column for which you want to aggregate the tokens
...  Specify the new aggregated columns in name-value pairs. The name is the name of the new column, and the value should be a function over a column in $tokens. For example: subject = paste(token, collapse = ' ') would create the column 'subject', of which the values are the concatenated tokens. See examples for more.
txt  If TRUE, add _txt column with concatenated tokens for by_label.
rm_by  If TRUE (default), remove the column(s) specified in by_label

Value

a transformed tCorpus

Examples

tc = tc_sotu_udpipe$copy()
tc$udpipe_clauses()

fold_rsyntax(tc, 'clause', by_label = 'subject', subject = paste(token, collapse= ' '))
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**

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

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

---

get_dtm

**Create a document term matrix.**

**Description**

Create a document term matrix. The default output is a sparse matrix (Matrix, dgTMatrix). Alternatively, the dtm style from the tm and quanteda package can be used.

The dfm function is shorthand for using quanteda’s dfm (document feature matrix) class. The meta data in the tcorpus is then automatically added as docvars in the dfm.
Usage

get_dtm(
  tc,
  feature,
  context_level = c("document", "sentence"),
  weight = c("termfreq", "docfreq", "tfidf", "norm_tfidf"),
  drop_empty_terms = T,
  form = c("Matrix", "tm_dtm", "quanteda_dfm"),
  subset_tokens = NULL,
  subset_meta = NULL,
  context = NULL,
  context_labels = T,
  feature_labels = T,
  ngrams = NA,
  ngram_before_subset = F
)

get_dfm(
  tc,
  feature,
  context_level = c("document", "sentence"),
  weight = c("termfreq", "docfreq", "tfidf", "norm_tfidf"),
  drop_empty_terms = T,
  subset_tokens = NULL,
  subset_meta = NULL,
  context = NULL,
  context_labels = T,
  feature_labels = T,
  ngrams = NA,
  ngram_before_subset = F
)

Arguments

tc : a tCorpus

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
get_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 iCorpus.
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.

Value

A document term matrix, in the format specified in the form argument

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

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

## alternatively, create quanteda ('quanteda_dfm') or tm ('tm_dtm') class for DTM
m = get_dtm(tc, 'feature', form = 'quanteda_dfm')
class(m)
m

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

## use weighting
m = get_dtm(tc, 'feature', weight = 'norm_tfidf')
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_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).
Usage

get_kwic(
  tc,
  hits = NULL,
  i = NULL,
  query = NULL,
  code = "",
  ntokens = 10,
  n = NA,
  nsample = NA,
  output_feature = "token",
  query_feature = "token",
  context_level = c("document", "sentence"),
  kw_tag = c("<", ">"),
  ...
)

Arguments

tc a tCorpus

hits results of feature search. see 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
search_features. If a query is given, then the ... argument is used to pass other
arguments to 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.

query_feature If query is used, the feature column that is used to perform the query

context_level Select the maximum 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 search_features for the query parameters
Details
This is mainly for viewing results in the R console. If you want to create a subset corpus based on the context of query results, you can use subset_query with the window argument. Also, the browse_hits function is a good alternative for viewing query hits in full text.

Examples

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

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

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

---

**get_stopwords**

Get a character vector of stopwords

**Description**
Get a character vector of stopwords

**Usage**

```r
get_stopwords(lang)
```

**Arguments**

- **lang**

**Value**
A character vector containing stopwords

**Examples**

```r
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))`

---

**melt_quanteda_dict**

*Convert a quanteda dictionary to a long data.table format*

**Description**

This is used internally in the tCorpus dictionary search functions, but can be used manually for more control. For example, adding numeric scores for sentiment dictionaries, and specifying which label/code to use in search_dictionary().

**Usage**

`melt_quanteda_dict(dict, column = "code", .index = NULL)`

**Arguments**

- `dict`: The quanteda dictionary
- `column`: The name of the column with the label/code. If dictionary contains multiple levels, additional columns are added with the suffix _l[i], where [i] is the level.
- `.index`: Do not use (used for recursive melting)
Value
A data.table

Examples

d = quanteda::data_dictionary_LSD2015
melt_quanteda_dict(d)

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 tCorpus objects are kept
keep_meta if 'intersect', then only the document meta columns that occur in all tCorpus 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 occurrence.
duplicate_tag a character string. if if_duplicate 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')

## with "rename", has 20 documents of which 5 duplicates
tc = merge_tcorpora(tc1, tc2, if_duplicate = 'rename')
tc$n_meta
sum(grepl('#D', tc$meta$doc_id))

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

---

plot.contextHits  

S3 plot for contextHits class

Description

S3 plot for contextHits class

Usage

## S3 method for class 'contextHits'
plot(x, min_weight = 0, backbone_alpha = NA, ...)

Arguments

x  
a contextHits object, as returned by search_contexts

min_weight  
Optionally, the minimum weight for an edge in the network

backbone_alpha  
Optionally, the alpha threshold for backbone extraction (similar to a p-value, and lower is more strict)

...  
not used

Examples

## Not run:
tc = create_tcorpus(sotu_texts, doc_column='id')
hits = search_contexts(tc, c('War# war* OR army OR bomb*','Terrorism# terroris*','Economy# econom* OR bank*','Education# educat* OR school*'))
plot(hits)
## End(Not run)

## End(Not run)

plot.featureAssociations

### Description

visualize feature associations

### Usage

```r
## S3 method for class 'featureAssociations'
plot(x, n = 25, size = c("chi2", "freq", "ratio"), ...)
```

### Arguments

- `x`: a `featureAssociations` object, created with the `feature_associations` function
- `n`: the number of words in the plot
- `size`: use "freq", "chi2" or "ratio" for determining the size of words
- `...`: additional arguments passed to `dtm_wordcloud`

### Examples

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

plot(comp, balance=TRUE)
plot(comp, mode = 'ratio_x')
plot(comp, mode = 'ratio_y')
```

plot.featureHits

### Description

S3 plot for `featureHits` class

### Usage

```r
## S3 method for class 'featureHits'
plot(x, min_weight = 0, backbone_alpha = NA, ...)
```
Arguments

x a featureHits object, as returned by search_features
min_weight Optionally, the minimum weight for an edge in the network
backbone_alpha Optionally, the alpha threshold for backbone extraction (similar to a p-value, and lower is more strict)
...

Examples

tc = create_tcorpus(sotu_texts, doc_column='id')
hits = search_features(tc, c("War# war* OR army OR bomb*", 'Terrorism# terroris*','Economy# econom* OR bank*','Education# educat* OR school*'))
plot(hits)

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. Whether a term is under- or overrepresented is indicated on the x-axis, which shows the log ratios (negative is underrepresented, positive is overrepresented). Use "ratio_x" or "ratio_y" to only plot overrepresented or underrepresented words using dtm_wordcloud
plot_semnet

**Description**

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

**Usage**

```r
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,
  ...
)
```

**Examples**

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

plot(comp, balance=TRUE)
plot(comp, mode = 'ratio_x')
plot(comp, mode = 'ratio_y')
```
Arguments

- **g**: A network in the igraph format. Specifically designed for the output of coOccurrenceNetwork() and windowedCoOccurrenceNetwork()
- **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 coOccurrenceNetwork functions to indicate the number of times a token occurred.
- **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)

g = semnet_window(tc, 'feature', window.size = 10)
g = backbone_filter(g, max_vertices = 100)
plot_semnet(g)

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 = 1,
  random.y = T,
  xlim = NULL,
  ylim = NULL,
  col = c("darkred", "navyblue"),
  fixed_col = 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
col                 A vector of colors that is passed to colorRamp to interpolate colors over x axis
fixed_col           Optionally, a vector of the exact colors given to words.
...                 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')

plot_words(x, y, words, c(1, 2, 3, 4))

---

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 = F,
  as_ascii = F,
  remove_punctuation = T,
  remove_stopwords = F,
  remove_numbers = F,
  min_freq = NULL,
  min_docfreq = NULL,
preprocess_tokens

max_freq = NULL,
max_docfreq = NULL,
min_char = NULL,
max_char = NULL,
ngram_skip_empty = T
)

Arguments

x A character or factor 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!)
remove_numbers remove features that are only numbers
min_freq an integer, specifying minimum token frequency.
min_docfreq an integer, specifying minimum document frequency.
max_freq an integer, specifying minimum token frequency.
max_docfreq an integer, specifying minimum document frequency.
min_char an integer, specifying minimum number of characters in a term
max_char an integer, specifying maximum number of characters in a term
ngram_skip_empty if ngrams are used, determines whether empty (filtered out) terms are skipped (i.e. c("this", NA, "test"), becomes "this_test") or

Value

a factor vector
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)

Description

S3 print for contextHits class

Usage

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

Arguments

x

a contextHits object, as returned by 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 = search_contexts(tc, c('query label# A AND B', 'second query# (A AND Q) OR ("D E") OR I'))
hits

Description

S3 print for featureHits class

Usage

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

Arguments

x

... 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 = search_contexts(tc, c('query label# A AND B', 'second query# (A AND Q) OR ("D E") OR I'))
hits
Arguments

x a featureHits object, as returned by search_features

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 = search_features(tc, c('query label# A AND B', 'second query# (A AND G) OR ("D E") OR I'))
hits

Description

S3 print for tCorpus class

Usage

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

Arguments

x a tCorpus object

Examples

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

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)

require_package(package, min_version = NULL)

Arguments

package The name of the package
min_version The minimum version

Value

An error if package does not exist
search_contexts  

*Search for documents or sentences using Boolean queries*

**Description**

Search for documents or sentences using Boolean queries

**Usage**

```r
search_contexts(
  tc, query, code = NULL,
  feature = "token",
  context_level = c("document", "sentence"),
  not = F,
  verbose = F,
  as_ascii = F
)
```

**Arguments**

- `tc` a `tCorpus`
- `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 specify the code argument, to label results.
- `code` If given, used as a label for the results of the query. Especially useful 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.
- `not` If TRUE, perform a NOT search. Return the articles/sentences for which the query is not found.
- `verbose` If TRUE, progress messages will be printed
- `as_ascii` if TRUE, perform search in ascii.

**Details**

Brief summary of the query language

The following operators and modifiers are supported:

- The standard 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 strictly 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 specifying the token distance. e.g. "climate change"~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

Value

A contextHits object, which is a list with $hits (data.frame with locations) and $queries (copy of queries for provenance)

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$tokens

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

## sentence level

hits = search_contexts(tc, 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

search_contexts(tc, 'A')$hits

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

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

```r
search_contexts(tc, 'A AND B')$hits
search_contexts(tc, 'A AND D')$hits
search_contexts(tc, 'A AND (B OR D)')$hits

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

## sequence search (adjacent words)

```r
search_contexts(tc, '"A B"')$hits
search_contexts(tc, '"A C"')$hits ## no hit, because not adjacent

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

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

## proximity search (using ~ flag)

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

search_contexts(tc, '"A (B OR D)"~5')$hits ## can contain nested OR
search_contexts(tc, '"<A <B C>"~5')$hits ## can contain nested sequence (must use <>)
## cannot contain nested AND or NOT!!
```

## case sensitive search

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

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

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

---

**search_dictionary**  

**Dictionary lookup**

---

**Description**

Similar to `search_features`, but for fast matching of large dictionaries.
**search_dictionary**

Usage

```r
search_dictionary(
  tc,
  dict,
  token_col = "token",
  string_col = "string",
  code_col = "code",
  sep = " ",
  mode = c("unique_hits", "features"),
  case_sensitive = F,
  use_wildcards = T,
  ascii = F,
  verbose = F
)
```

Arguments

- **tc**
  A tCorpus

- **dict**
  A dictionary. Can be either a data.frame or a quanteda dictionary. If a data.frame is given, it has to have a column named "string" (or use string_col argument) that contains the dictionary terms, and a column "code" (or use code_col argument) that contains the label/code represented by this string. Each row has a single string, that can be a single word or a sequence of words separated by a whitespace (e.g., "not bad"), and can have the common ? and * wildcards. If a quanteda dictionary is given, it is automatically converted to this type of data.frame with the `melt_quanteda_dict` function. This can be done manually for more control over labels.

- **token_col**
  The feature in tc that contains the token text.

- **string_col**
  If dict is a data.frame, the name of the column in dict with the dictionary lookup string. Default is "string"

- **code_col**
  The name of the column in dict with the dictionary code/label. Default is "code". If dict is a quanteda dictionary with multiple levels, "code_l2", "code_l3", etc. can be used to select levels.

- **sep**
  A regular expression for separating multi-word lookup strings (default is " ", which is what quanteda dictionaries use). For example, if the dictionary contains "Barack Obama", sep should be " " so that it matches the consecutive tokens "Barack" and "Obama". In some dictionaries, however, it might say "Barack+Obama", so in that case sep = '\+' should be used.

- **mode**
  There are two modes: "unique_hits" and "features". The "unique_hits" mode prioritizes finding unique matches, which is recommended for counting how often a dictionary term occurs. If a term matches multiple dictionary terms (which should only happen for nested multi-word terms, such as "bad" and "not bad"), the longest term is always used. The features mode does not delete duplicates.

- **case_sensitive**
  Logical, should lookup be case sensitive?

- **use_wildcards**
  Use the wildcards * (any number including none of any character) and ? (one or none of any character). If FALSE, exact string matching is used.
search_features

ascii  If true, convert text to ascii before matching
verbose  If true, report progress

Value

A vector with the id value (taken from dict$id) for each row in tc$tokens

Examples

dict = data.frame(string = c('this is', 'for a', 'not big enough'), code=c('a','c','b'))
tc = create_tcorpus(c('this is a test', 'This town is not big enough for a test'))
search_dict$tc$tokens

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.

Usage

search_features(
  tc,
  query,
  code = NULL,
  feature = "token",
  mode = c("unique_hits", "features"),
  context_level = c("document", "sentence"),
  keep_longest = TRUE,
  as_ascii = F,
  verbose = F
)

Arguments

tc  a tCorpus

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 specify 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.
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).

Select whether the queries should occur within while "documents" or specific "sentences".

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".

If TRUE, perform search in ascii.

If TRUE, progress messages will be printed

Brief summary of the query language

The following operators and modifiers are supported:

- The standard 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 strictly 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 specifying the token distance. e.g. "climate chang*e"~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 useful 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.

Value

A featureHits object, which is a list with $hits (data.frame with locations) and $queries (copy of queries for provenance)

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$tokens ## (example uses letters instead of words for simple query examples)

hits = search_features(tc, c('A AND B', 'second query# (A AND Q) OR ("D E") OR I'))
hits$hits ## print shows number of hits

hits = search_features(tc, 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
search_features(tc, 'A')$hits
search_features(tc, 'G*')$hits ## wildcard *
search_features(tc, 'G?')$hits ## wildcard *
search_features(tc, 'GG')$hits

# boolean
search_features(tc, 'A AND B')$hits
search_features(tc, 'A AND D')$hits
search_features(tc, 'A AND (B OR D)')$hits
search_features(tc, 'A NOT B')$hits
search_features(tc, 'A NOT (B OR D)')$hits

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

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

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

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

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

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

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

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

## ghost terms (~g flag)
search_features(tc, 'A AND B~g')$hits ## ghost term (~g) has to occur, but is not returned
search_features(tc, 'A AND Q~g')$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')

search_features(tc, '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)

search_features(tc, '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
search_features(tc, 'A AND B~g')$hits
### Description

This function calculates the co-occurrence of features and returns a network/graph in the igraph format, where nodes are tokens and edges represent the similarity/adjacency of tokens. Co-occurrence is calculated based on how often two tokens occurred 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.

### Usage

```r
semnet(
  tc,
  feature = "token",
  measure = c("con_prob", "con_prob_weighted", "cosine", "count_directed",
  "count_undirected", "chi2"),
  context_level = c("document", "sentence"),
  backbone = F,
  n.batches = NA
)
```

### Arguments

- **tc**: a `tCorpus` or a `featureHits` object (i.e. the result of `search_features`)
- **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-occur 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

### Value

an `igraph` graph in which nodes are features and edges are similarity scores

### Examples

```r
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 = semnet(tc, 'token')
```
semnet_window

Create a semantic network based on the co-occurrence of tokens in token windows

Description

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

If a featureHits object is given as input, then for for query hits that have multiple positions (i.e. terms connected with AND statements or word proximity) the raw count score is biased. For the count_* measures therefore only the first position of the query hit is used.

Usage

semnet_window(
  tc,
  feature = "token",
  measure = c("con_prob", "cosine", "count_directed", "count_undirected", "chi2"),
  context_level = c("document", "sentence"),
  window.size = 10,
  direction = "<>",
  backbone = F,
  n.batches = 5,
  matrix_mode = c("positionXwindow", "windowXwindow")
)

Arguments

tc a tCorpus or a featureHits object (i.e. the result of search_features)
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-occur within "documents" or "sentences"
window.size The token distance within which features are considered to co-occur
direction Determine whether co-occurrence is asymmetric ("<>") 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
set_network_attributes

n.batches
To limit memory use the calculation is divided into batches. This parameter controls the number of batches.

matrix_mode
There are two approaches for calculating window co-occurrence (see details). By default we use positionXmatrix, but matrixXmatrix is optional because it might be favourable for some uses, and might make more sense for cosine similarity.

Details
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, which we call the "windowXwindow" mode. The positionXwindow approach has the advantage of being easy to interpret (e.g. how likely is feature "Y" to occur within 10 tokens from feature "X"?). The windowXwindow mode, on the other hand, has the interesting feature that similarity is stronger if tokens co-occur more closely together (since then their windows overlap more), but this only works well for similarity measures that normalize the similarity (e.g., cosine). Currently, we only use the positionXwindow mode, but windowXwindow could be interesting to use as well, and for cosine it might actually make more sense.

Value
an Igraph graph in which nodes are features and edges are similarity scores

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 = semnet_window(tc, 'token', window.size = 1)
g
igraph::get.data.frame(g)
plot_semnet(g)

---

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

```r
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

```r
tc = create_tcorpus(c("Var A B C", "Var B C", "Var B D"))
g = semnet(tc, "token")

igraph::get.edge.attribute(g)
igraph::get.vertex.attribute(g)
plot(g)

g = set_network_attributes(g, size_attribute = "freq")
igraph::get.edge.attribute(g)
igraph::get.vertex.attribute(g)
plot(g)
```

Description

Usage

gt(freq)

Arguments

freq A numeric vector of frequencies (integers).

Value

A numeric vector with the smoothed term proportions

Description

Returns a data.table with the language, treebank and udpipe_model name. Uses the default model repository provided by the udpipe package (udpipe_download_model). For more information about udpipe and performance benchmarks of the UD models, see the GitHub page of the udpipe package.

Usage

show_udpipe_models()

Value

a data.frame

Examples

show_udpipe_models()

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

subset.tCorpus  

S3 subset for tCorpus class

Description

S3 subset for tCorpus class

Usage

## S3 method for class 'tCorpus'
subset(x, subset = NULL, subset_meta = NULL, window = NULL, ...)

Arguments

x  
a tCorpus object

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 specifying 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.

...  
not used
Examples

```r
## create tcorpus of 5 bush and obama docs
tc = create_tcorpus(sotu_texts[c(1:5,801:805),], doc_col='id')

## subset to keep only tokens where token_id <= 20 (i.e. first 20 tokens)
tcs1 = subset(tc, token_id < 20)
tcs1

## subset to keep only documents where president is Barack Obama
subset_meta = president == 'Barack Obama')
tcs2 = subset(tc, subset_meta = president == 'Barack Obama')
tcs2
```

---

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.

Usage

```r
subset_query(
  tc,  # A tCorpus
  query,  # A character string that is a query. See search_contexts for query syntax.
  feature = "token",  # The name of the feature columns on which the query is used.
  context_level = c("document", "sentence"),  # Select whether the query and subset are performed at the document or sentence level.
  not = F,  # If TRUE, perform a NOT search. Return the articles/sentences for which the query is not found.
  as_ascii = F,  # if TRUE, perform search in ascii.
  window = NA  # If used, uses a word distance as the context (overrides context_level)
)
```

Arguments

- `tc` A tCorpus
- `query` A character string that is a query. See 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 or sentence level.
- `not` If TRUE, perform a NOT search. Return the articles/sentences for which the query is not found.
- `as_ascii` If TRUE, perform search in ascii.
- `window` If used, uses a word distance as the context (overrides context_level)

Details

See the documentation for search_contexts for an explanation of the query language.
**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
tc2 = subset_query(tc, 'A')
tc2$meta

 summary.contextHits  

**Description**

S3 summary for contextHits class

**Usage**

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

**Arguments**

object  
a contextHits object, as returned by `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 = search_contexts(tc, c('query label# A AND B', 'second query# (A AND Q) OR ("D E") OR I'))

summary(hits)

 summary.featureHits  

**Description**

S3 summary for featureHits class

**Usage**

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

**Arguments**

object  
a featureHits object, as returned by `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 = search_features(tc, c('query label# A AND B', 'second query# (A AND Q) OR ("D E") OR I'))

summary(hits)
Arguments

object a featureHits object, as returned by 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 = search_features(tc, 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

The tCorpus is a class for managing tokenized texts, stored as a data.frame in which each row represents a token, and columns contain the positions and features of these tokens.
Methods and Functions

The corpustools package uses both functions and methods for working with the tCorpus. Methods are used for all operations that modify the tCorpus itself, such as subsetting or adding columns. This allows the data to be modified by reference. Methods are accessed using the dollar sign after the tCorpus object. For example, if the tCorpus is named tc, the subset method can be called as tc$subset(...)

Functions are used for all operations that return a certain output, such as search results or a semantic network. These are used in the common R style that you know and love. For example, if the tCorpus is named tc, a semantic network can be created with semnet(tc, ...)

Overview of methods and functions

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.

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

---

**tCorpus$annotate_rsyntax**

*Annotate tokens based on rsyntax queries*

**Description**

Apply queries to extract syntax patterns, and add the results as three columns to a tokenlist. The first column contains the ids for each hit. The second column contains the annotation label. The third column contains the fill level (which you probably won’t use, but is important for some features). Only nodes that are given a name in the tquery (using the label parameter) will be added as annotation.

Note that while queries only find 1 node for each labeled component of a pattern (e.g., quote queries have 1 node for "source" and 1 node for "quote"), all children of these nodes can be annotated by setting fill to TRUE. If a child has multiple ancestors, only the most direct ancestors are used (see documentation for the fill argument).

**Usage:**

```r
## R6 method for class tCorpus. Use as tc$method (where tc is a tCorpus object).
annotate_rsyntax(column, ..., block = NULL, fill = TRUE, overwrite = FALSE, block_fill = FALSE, copy = TRUE, verbose = FALSE)
```
Arguments

  column  The name of the column in which the annotations are added. The unique ids are added as column_id

  ...  One or multiple tqueries, or a list of queries, as created with tquery. Queries can be given a named by using a named argument, which will be used in the annotation_id to keep track of which query was used.

  block  Optionally, specify ids (doc_id - sentence - token_id triples) that are blocked from querying and filling (ignoring the id and recursive searches through the id).

  fill  Logical. If TRUE (default) also assign the fill nodes (as specified in the tquery). Otherwise these are ignored

  overwrite  Applies if column already exists. If TRUE, existing column will be overwritten. If FALSE, the existing annotations in the column will be blocked, and new annotations will be added. This is identical to using multiple queries.

  block_fill  If TRUE (and overwrite is FALSE), the existing fill nodes will also be blocked. In other words, the new annotations will only be added if the

  verbose  If TRUE, report progress (only useful if multiple queries are given)

Examples

library(rsyntax)

## spacy tokens for: Mary loves John, and Mary was loved by John
tokens = tokens_spacy[tokens_spacy$doc_id == 'text3',]
tc = tokens_to_tcorpus(tokens)

## two simple example tqueries
passive = tquery(pos = "VERB*", label = "predicate",
  children(relation = c("agent"), label = "subject"))
active = tquery(pos = "VERB*", label = "predicate",
  children(relation = c("nsubj", "nsubjpass"), label = "subject"))

tc$annotate_rsyntax("clause", pas=passive, act=active)
tc$tokens

if (interactive()) {
  plot_tree(tc$tokens, annotation='clause')
}
if (interactive()) {
syntax_reader(tc$tokens, annotation = 'clause', value='subject')
}
Description

Add a column to the token data that contains a code (the query label) for tokens that match the dictionary

Usage:

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

code_dictionary(...)```

Arguments

dict A dictionary. Can be either a data.frame or a quanteda dictionary. If a data.frame is given, it has to have a column named "string" (or use string_col argument) that contains the dictionary terms. All other columns are added to the tCorpus $tokens data. Each row has a single string, that can be a single word or a sequence of words seperated by a whitespace (e.g., "not bad"), and can have the common ? and * wildcards. If a quanteda dictionary is given, it is automatically converted to this type of data.frame with the `melt_quanteda_dict` function. This can be done manually for more control over labels.

token_col The feature in tc that contains the token text.

string_col If dict is a data.frame, the name of the column in dict that contains the dictionary lookup string

sep A regular expression for separating multi-word lookup strings (default is " ", which is what quanteda dictionaries use). For example, if the dictionary contains "Barack Obama", sep should be " " so that it matches the conseguitive tokens "Barack" and "Obama". In some dictionaries, however, it might say "Barack+Obama", so in that case sep = '\+' should be used.

case_sensitive logical, should lookup be case sensitive?

column The name of the column added to $tokens. [column]_id contains the unique id of the match. If a dictionary has multiple levels, these are added as [column]_l[level].

use_wildcards Use the wildcards * (any number including none of any character) and ? (one or none of any character). If FALSE, exact string matching is used. (":-)" versus ":; -: ”)”. This is only behind the scenes for the dictionary lookup, and will not affect tokenization in the corpus.

ascii If true, convert text to ascii before matching

verbose If true, report progress

Value

the tCorpus
Examples

dict = data.frame(string = c("good", "bad", "ugly", "nice", "not pretty", ":)", 
                   sentiment=c(1,-1,-1,1,-1,1))

tc = create_tcorpus(c("The good, the bad and the ugly, is nice :) but not pretty :(")

tc$code_dictionary(dict)

tc$tokens

tCorpus$code_features

Description

like search_features, but instead of return hits only adds a column to the token data that contains 
a code (the query label) for tokens that match the query. Note that only one code can be assigned to 
each token, so if there are overlapping results for different queries, the code for the last query will 
be used. This means that the order of queries (in the query argument) matters.

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', ...)

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 "documents" or specific 
"sentences".

as_ascii if TRUE, perform search in ascii.

verbose If TRUE, progress messages will be printed

overwrite If TRUE (default) and column already exists, overwrite previous results.

... alternative way to specify name-value pairs for adding additional columns

Examples

tc = create_tcorpus("Anna and Bob are secretive")

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

**Details**

**Usage:**

```r
## 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**

```r
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. 

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

**Usage:**

```r
## 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_docfreq_pct=0.5, measure=c('cosine','overlap_pct'), similarity=1, keep=c('first','last','random'), weight=c('norm_tfidf','tfidf','termfreq','docfreq'), ngrams=NA, print_duplicates=F, copy=F)
```
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 
A vector of length 1 or 2. If length is 1, the same value is used for the left and right side of the window. If length is 2, the first and second value determine the left and right side. For example, the value 12 will compare each document to all documents between the previous and next 12 hours, and c(-10, 36) will compare each document to all documents between the previous 10 and the next 36 hours.

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

max_docfreq_pct 
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),

lowercase 
If True, make feature lowercase

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

verbose 
if TRUE, report progress

copy 
If TRUE, the method returns a new tCorpus object instead of deduplicating the current one by reference.

Examples

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

Delete column from the data and meta data

### Description

**Usage:**

#### Arguments

- `cnames`: the names of the columns to delete

### Details

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

delete_columns(cnames)
```

### Examples

```r
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)

# Delete column from the data

# Data

tc$tokens
tc$delete_columns('token')
tc$tokens

# Meta

tc$meta
tc$delete_meta_columns('date')
tc$meta
```
tCorpus$feats_to_columns

Cast the "feats" column in UDpipe tokens to columns

Description

If the UDpipe parser is used in create_tcorpus, the 'feats' column contains strings with features (e.g., Number=Sing|PronType=Dem). To work with these nested features it is more convenient to cast them to columns.

Arguments

- **keep** 
  Optionally, the names of features to keep
- **drop** 
  Optionally, the names of features to drop
- **rm_column** 
  If TRUE (default), remove the original column

Details

Usage:

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

feats_to_columns(keep=NULL, drop=NULL, rm_column=TRUE)
```

Examples

```r
if (interactive()) {
  tc = create_tcorpus('This is a test Bobby.', udpipe_model='english-ewt')
  tc$feats_to_columns()
  tc$tokens

  tc = create_tcorpus('This is a test Bobby.', udpipe_model='english-ewt')
  tc$feats_to_columns(keep = c('Gender', 'Tense', 'Person'))
  tc$tokens
}
```

---

tCorpus$feature_subset

*Filter features*
Description

Similar to using \texttt{tCorpus$subset}, but instead of deleting rows it only sets rows for a specified feature to \texttt{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 \texttt{subset}, it is easy to use objects and functions in the filter, including the special functions for using term frequency statistics (see documentation for \texttt{tCorpus$subset}).

Usage:

\texttt{## R6 method for class \texttt{tCorpus}. Use as tc$method (where tc is a tCorpus object).

feature_subset(column, new_column, subset)}

Arguments

- \texttt{column} the column containing the feature to be used as the input
- \texttt{subset} logical expression indicating rows to keep in the tokens data. i.e. rows for which the logical expression is \texttt{FALSE} will be set to \texttt{NA}.
- \texttt{new_column} the column to save the filtered feature. Can be a new column or overwrite an existing one.
- \texttt{min_freq} an integer, specifying minimum token frequency.
- \texttt{min_docfreq} an integer, specifying minimum document frequency.
- \texttt{max_freq} an integer, specifying minimum token frequency.
- \texttt{max_docfreq} an integer, specifying minimum document frequency.
- \texttt{min_char} an integer, specifying minimum characters in a token
- \texttt{max_char} an integer, specifying maximum characters in a token

Examples

\begin{verbatim}
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$tokens
\end{verbatim}


---

\texttt{tCorpus$fold_rsyntax}  \textit{Fold rsyntax annotations}

Description

If a \texttt{tCorpus} has rsyntax annotations (see \texttt{annotate_rsyntax}), it can be convenient to aggregate tokens that have a certain semantic label. For example, if you have a query for labeling "source" and "quote", you can add an aggregated value for the sources (such as a unique ID) as a column, and then remove the quote tokens.
Arguments

- `annotation`: The name of an rsyntax annotation column
- `by_label`: The labels in this column for which you want to aggregate the tokens
- `txt`: If TRUE, add _txt column with concatenated tokens for by_label
- `rm_by`: If TRUE (default), remove the column(s) specified in by_label
- `copy`: If TRUE, return a copy of the transformed tCorpus, instead of transforming the tCorpus by reference

Details

Usage:

```r
fold_rsyntax((annotation, by_label, ..., to_label=NULL, rm_by=T, copy=F)
```

Examples

```r
tc = tc_sotu_udpipe$copy()
tc$udpipe_clauses()
tc$fold_rsyntax('clause', by_label = 'subject', subject = paste(token, collapse=' '))
tc$tokens
```

Description

Get (a copy of) the token and meta data. For quick access recommend using tc$tokens and tc$meta to get the tokens and meta data.tables, which does not copy the data. However, you should then make sure to not change the data.tables by reference, or you might break the tCorpus.

Usage:

```r
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)
```
tCorpus$get

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$subset method)
- **doc_id**: A vector with document ids to select rows. Faster than subset, because it uses binary search. Cannot be used in combination with subset. If duplicate doc_ids are given, duplicate rows are returned.
- **token_id**: A vector with token indices. Can only be used in pairs with doc_id. For example, if doc_id = c(1,1,1,2,2) and token_id = c(1,2,3,1,2), then the first three tokens of doc 1 and the first 2 tokens of doc 2 are returned. This is mainly useful for fast (binary search) retrieval of specific tokens.
- **safe_copy**: for advanced use. The get methods always return a copy of the data, even if the full data is returned (i.e. use get without parameters). This is to prevent accidental changes within tCorpus data (which can break it) if the returned data is modified by reference (see data.table documentation). If safe_copy is set to FALSE and get is called without parameters—t$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.

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$tokens           ## full data.table
tc$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$meta

## option to repeat meta data to match tokens
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.

### Usage:

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

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

### 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)

### Examples

```r
if (interactive()) {
  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$tokens
}
```
**tCorpus$merge**

Merge the token and meta data.tables of a tCorpus with another data.frame

---

**Description**

Add columns to token/meta by merging with a data.frame df. Only possible for unique matches (i.e. the columns specified in by are unique in df)

**Arguments**

- **df**
  A data.frame (can be regular, data.table or tibble)
- **by**
  The columns to match on. Must exist in both tokens/meta and df. If the columns in tokens/meta and df have different names, use by.x and by.y
- **by.x**
  The names of the columns used in tokens/meta
- **by.y**
  The names of the columns used in df
- **columns**
  Optionally, specify which specific columns from df to merge to tokens

**Details**

**Usage:**

```r
## R6 method for class tCorpus. Use as tc$method (where tc is a tCorpus object).
merge(df, by, by.x, by.y)
merge_meta(df, by, by.x, by.y)
```

**Examples**

```r
d = data.frame(text = c('This is an example. Best example ever.', 'oh my god', 'so good'),
  id = c('a', 'b', 'c'),
  source = c('aa', 'bb', 'cc'))
tc = create_tcorpus(d, doc_col = 'id', split_sentences = TRUE)

df = data.frame(doc_id=c('a', 'b'), test=c('A', 'B'))
tc$merge(df, by='doc_id')
tc$tokens

df = data.frame(doc_id=c('a', 'b'), sentence=1, test2=c('A', 'B'))
tc$merge(df, by=c('doc_id', 'sentence'))
tc$tokens

df = data.frame(doc_id=c('a', 'b'), sentence=1, token_id=c(3,4), test3=c('A', 'B'))
tc$merge(df, by=c('doc_id', 'sentence', 'token_id'))
tc$tokens
meta = data.frame(doc_id=c('a', 'b'), test=c('A', 'B'))
```
tc$merge_meta(meta, by='doc_id')
tc$meta

meta = data.frame(source=c('aa'), test2=c('A'))
tc$merge_meta(meta, by='source')
tc$meta

### Description

#### Usage:

#### 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 useful 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.
- **remove_numbers**
  - remove features that are only numbers
- **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.
- **max_freq**
  - an integer, specifying minimum token frequency.
- **max_docfreq**
  - an integer, specifying minimum document frequency.
- **min_char**
  - an integer, specifying minimum number of characters in a term
- **max_char**
  - an integer, specifying maximum number of characters in a term
tCorpus$replace_dictionary

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

```r
preprocess(column='token', new_column='feature', lowercase=T, ngrams=1,
ngram_context=c('document', 'sentence'), as_ascii=F, remove_punctuation=T,
remove_stopwords=F, remove_numbers=F, use_stemming=F, language='english',
min_freq=NULL, min_docfreq=NULL, max_freq=NULL, max_docfreq=NULL, min_char=NULL, max_char=NULL)
```

### Examples

```r
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$preprocess('token', 'preprocessed_4', ngrams = 3)
tc$tokens
tc$tokens
```

---

tCorpus$replace_dictionary

*Replace tokens with dictionary match*

### Description

Uses `search_dictionary`, and replaces tokens that match the dictionary lookup term with the dictionary code. Multi-token matches (e.g., "Barack Obama") will become single tokens. Multiple lookup terms per code can be used to deal with alternatives such as "Barack Obama", "president Obama" and "Obama".

This method can also be used to concatenate ASCII symbols into emoticons, given a dictionary of emoticons.

### Usage:

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

replace_dictionary(...)```
Arguments

dict  A dictionary. Can be either a data.frame or a quanteda dictionary. If a data.frame is given, it has to have a column named "string" (or use string_col argument) that contains the dictionary terms, and a column "code" (or use code_col argument) that contains the label/code represented by this string. Each row has a single string, that can be a single word or a sequence of words separated by whitespace (e.g., "not bad"), and can have the common ? and * wildcards. If a quanteda dictionary is given, it is automatically converted to this type of data.frame with the melt_quanteda_dict function. This can be done manually for more control over labels. Finally, you can also just pass a character vector. All multi word strings (like emoticons) will then be collapsed into single tokens.

token_col  The feature in tc that contains the token text.

string_col  If dict is a data.frame, the name of the column in dict with the dictionary lookup string. Default is "string"

code_col  The name of the column in dict with the dictionary code/label. Default is "code". If dict is a quanteda dictionary with multiple levels, "code_l2", "code_l3", etc. can be used to select levels.

replace_cols  The names of the columns in tc$tokens that will be replaced by the dictionary code. Default is the column on which the dictionary is applied, but in some cases it might make sense to replace multiple columns (like token and lemma)

sep  A regular expression for separating multi-word lookup strings (default is " ", which is what quanteda dictionaries use). For example, if the dictionary contains "Barack Obama", sep should be " " so that it matches the consecutive tokens "Barack" and "Obama". In some dictionaries, however, it might say "Barack+Obama", so in that case sep = '\+' should be used.

code_from_features  If TRUE, instead of replacing features with the matched code columnm, use the most frequent occuring string in the features.

code_sep  If code_from_features is TRUE, the separator for pasting features together. Default is an underscore, which is recommended because it has special features in corpustools. Most importantly, if a query or dictionary search is performed, multi-word tokens concatenated with an underscore are treated as separate consecutive words. So, "Bob_Smith" would still match a lookup for the two consecutive words "bob smith"

decrement_ids  If TRUE (default), decrement token ids after concatenating multi-token matches. So, if the tokens c(";", ")", "yay") have token_id c(1,2,3), then after concatenating ASCII emoticons, the tokens will be c( "):", "yay") with token_id c(1,2)

case_sensitive  logical, should lookup be case sensitive?

use_wildcards  Use the wildcards * (any number including none of any character) and ? (one or none of any character). If FALSE, exact string matching is used

ascii  If true, convert text to ascii before matching

verbose  If true, report progress

Value

A vector with the id value (taken from dict$id) for each row in tc$tokens
Examples

tc = create_tcorpus('happy :) sad :( happy 8-')
tc$tokens  ## tokenization has broken up emoticons (as it should)

# corpustools dictionary lookup automatically normalizes tokenization of
# tokens and dictionary strings. The dictionary string ";)
# matching the single token ";)" and two consecutive tokens c(";", ")
# makes it easy and foolproof to look for emoticons like this:
emoticon_dict = data.frame(
  code = c('happy_emo', 'happy_emo', 'sad_emo'),
  string = c(':', '8-', ':('))
tc$replace_dictionary(emoticon_dict)
tc$tokens

# If a string is passed to replace dictionary, it will collapse multi-word
# strings... .
tc = create_tcorpus('happy :) sad :( Barack Obama')
tc$tokens
tc$replace_dictionary(c(';', '8-', 'Barack Obama'))
tc$tokens

---

**tCorpus$search_recode**

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

**Description**

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

**Usage:**

```r
## 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 = FALSE)
```

**Arguments**

- `feature` The feature in which to search
- `new_value` the character string with which all features that are found are replaced
- `query` See `search_features` for the query parameters
- `...` Additional search_features parameters. See `search_features`
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 while 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
subset_value If subset is used, should value also be subsetted? Default is TRUE, which is what you want if the value has the same length as the full data.table (which is the case if a column in tokens is used). However, if the vector of values is already of the length of the subset, subset_value should be FALSE

Details

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[1:5,], doc_column = 'id')

tc$tokens ## 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$set('second_token', token, subset = token_id == 2)
tCorpus$set_levels

Change levels of factor columns

Description

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

Arguments

column the name of the column
levels The new levels

Details

Usage:

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

set_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$tokens
tCorpus$set_name  Change column names of data and meta data

Description

Usage:

Arguments

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

Details

## 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[1:5,,], doc_column = 'id')

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

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

There are two flavours. You can either use subset(tc, ...) or tc$subset(...). The difference is that the second approach changes the tCorpus by reference. In other words, tc$subset() will delete the rows from the tCorpus, instead of creating a new tCorpus. Modifying the tCorpus by reference is more efficient (which becomes important if the tCorpus is large), but the more classic subset(tc, ...) approach is often more obvious.

Subset can also be used to select rows based on token/feature frequencies. 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 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.

Usage:

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

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

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 specifying 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)

Examples

```r
tc = create_tcorpus(sotu_texts[1:5,], doc_column = 'id')
tc$n # original number of tokens

## select only first 20 tokens per document
tc2 = subset(tc, token_id < 20)
tc2$n
```
## Note that the original is untouched

tc$n

## Now we subset by reference. This doesn't make a copy, but changes tc itself

tc$subset(token_id < 20)
tc$n

## you can filter on term frequency and document frequency with the freq_filter() and
docfreq_filter() functions

tc = create_tcorpus(sotu_texts[c(1:5,800:805),], doc_column = 'id')
tc$subset( freq_filter(token, min = 2, max = 4) )
tc$tokens

#### subset can be used for meta data by using the subset_meta argument, or the subset_meta method

tc$n_meta
tc$meta
tc$subset(subset_meta = president == 'Barack Obama')
tc$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 `search_contexts` for an explanation of the query language.

### Usage:

```r
## 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','window'))
```

### Arguments

- **query**
  
  A character string that is a query. See `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 or sentence
  level.

- **window**
  
  If used, uses a word distance as the context (overrides context_level)

- **as_ascii**
  
  if TRUE, perform search in ascii.

- **not**
  
  If TRUE, perform a NOT search. Return the articles/sentences for which the
  query is not found.

- **copy**
  
  If TRUE, return modified copy of data instead of subsetting the input tcorpus by
  reference.
tCorpus$udpipe_clauses

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$subset_query('A')
tc$meta

## using copy mechanic
class(tc$tokens$doc_id)
tc2 = tc$subset_query('A AND D', copy=TRUE)
tc2$get_meta()
tc$meta ## (unchanged)

tCorpus$udpipe_clauses

Add columns indicating who did what

Description

An off-the-shelf application of rsyntax for extracting subject-verb clauses. Designed for working with a tCorpus created with udpipe_tcorpus.

Arguments

column The name of the column in $tokens to store the results.
tqueries A list of tQueries. By default uses the off-the-shelf tqueries in udpipe_clause_tqueries

Value

a tCorpus

Examples

tc = tc_sotu_udpipe$copy()
tc$udpipe_clauses()
if (interactive()) {
  tc_plot_tree(tc, token, lemma, POS, annotation='clause')
browse_texts(tc, rsyntax='clause', value='subject')
}
tCorpus$udpipe_coref  Perform rule based coreference resolution

Description

This is an experimental implementation of coreference resolution, aimed at coreferences of nouns and pronouns.

Arguments

keep_feats  If TRUE, keep the feature columns created for the coref resolution
rm_unique  If TRUE, remove unique coref ids (i.e. if )

Value

a tCorpus

Examples

tc = tc_sotu_udpipe$copy()
tc$udpipe_quotes()
## Not run:
tc$udpipe_coref()
tc$tokens[!is.na(tc$tokens$coref_id),]
## End(Not run)

---

tCorpus$udpipe_quotes  Add columns indicating who said what

Description

An off-the-shelf application of rsyntax for extracting quotes. Designed for working with a tCorpus created with udpipe_tcorpus.

Arguments

tqueries  A list of tqueries. By default uses the off-the-shelf tqueries in udpipe_quote_tqueries.
span_tqueries  Additional tqueries for finding candidates for 'span quotes' (i.e. quotes that span multiple sentences, indicated by quotation marks). By default uses the off-the-shelf tqueries in udpipe_spanquote_tqueries.
Details

Default tqueries are provided for detecting source-quote relations within sentences (udpipe_quote_tqueries), and for detecting source candidates for text between quotation marks that can span across multiple sentences (udpipe_spanquote_tqueries). These have mainly been developed and tested for the english-ewt udpipe model.

There are two ways to customize this function. One is to specify a custom character vector of verb lemma. This vector should then be passed as an argument to the two functions for generating the default tqueries. The second (more advanced) way is to provide a custom list of tqueries. (Note that the udpipe_quote_tqueries and udpipe_spanquote_tqueries functions simply create lists of queries.

You can create new lists, or add tqueries to these lists). If you create custom tqueries, make sure that the labels for the quote and source tokens are 'source' and 'quote'. For the spanquote_tqueries, the label for the source candidate should be 'source'.

Value

the columns 'quote', 'quote_id', and 'quote_verbatim' are added to tokens

Examples

## Not run:

```r
txt = 'Bob said that he likes Mary. John did not like that:
    "how dare he!". "It is I, John, who likes Mary!!"'
tc = udpipe_tcorpus(txt, model = 'english-ewt')
tc$udpipe_quotes()
if (interactive()) {
  tc_plot_tree(tc, token, lemma, POS, annotation='quote')
  browse_texts(tc, rsyntax='quote', value='source')
}
## you can provide your own lists of tqueries, or use the two
## query generating functions to customize the specific 'verb lemma'
## (i.e. the lemma for verbs that indicate speech)
custom_verb_lemma = c('say', 'state')  ## this should be longer
quote_tqueries = udpipe_quote_tqueries(custom_verb_lemma)
span_quote_tqueries = udpipe_spanquote_tqueries(custom_verb_lemma)

## note that these use simply lists with tqueries, so you can also
## create your own list or customize these lists

quote_tqueries
span_quote_tqueries

if (interactive()) {
  tc$udpipe_quotes(tqueries = quote_tqueries, span_tqueries = span_quote_tqueries)
  tc_plot_tree(tc, token, lemma, POS, annotation='quote')
  browse_texts(tc, rsyntax='quote', value='source')
}
## End(Not run)
### 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 and functions for viewing, modifying and subsetting tCorpus data**

**Description**

(back to overview)

**Details**

**Get data**

- **$get()** Get (by default deep copy) token data, with the possibility to select columns and subset. Instead of copying you can also access the token data with tc$tokens.
- **$get_meta()** Get meta data, with the possibility to select columns and subset. Like tokens, you can also access meta data with tc$meta.
- **get_dtm()** Create a document term matrix
get_dfm() Create a document term matrix, using the Quanteda dfm format
$status() 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 subset function. A subset expression can be specified for both the token data (subset) and the document meta data (subset_meta).
subset_query() Subset the tCorpus based on a query, as used in search_contexts
$subset() Like subset, but as an R6 method that changes the tCorpus by reference
$subset_query() Like subset_query, but as an R6 method that changes the tCorpus by reference

**Fields**

For the sake of convenience, the number of rows and column names of the data and meta data.tables can be accessed directly.

$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)*
tCorpus_modify_by_reference

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

$update() 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 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 slower and less memory efficient.

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!!

Using the R6 method for subset forces this approach on you, because it is faster and more memory efficient. If you do want to make a copy, there are several solutions.

Firstly, for some methods we provide identical functions. For example, instead of the $subset() R6 method, we can use the subset() function.

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

We promise that only the R6 methods (called as tc$method()) will change the data by reference.

A second option is that R6 methods where copying is often useful have copy parameter Modifying by reference only happens in the R6 methods

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

Finally, you can always make a deep copy of the entire tCorpus before modifying it, using the $copy() method.

tc2 = tc$copy()

---

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
browse_hits() Create full-text browsers with highlighted search hits

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

---

tc_plot_tree  
*Visualize a dependency tree*

**Description**

A wrapper for the `plot_tree` function, that can be used directly on a tCorpus.

**Usage**

```
tc_plot_tree()
```
Arguments

tc    a tCorpus
...
Arguments passed to plot_tree. Most importantly, this is used to select which
specific columns to display on the bottom rows. For instance, tc_plot_tree(tc,
token, lemma, POS) shows only these three columns.

annotation    Optionally, the name of a column with an rsyntax annotation.

sentence_i    By default, plot_tree uses the first sentence (sentence_i = 1) in the data. sen-
tence_i can be changed to select other sentences by position (the i-th unique
sentence in the data). Note that sentence_i does not refer to the values in the
sentence column (for this use the sentence argument together with doc_id)

doc_id    Optionally, the document id can be specified. If so, sentence_i refers to the i-th
sentence within the given document.

pdf_file    Directly save the plot as a pdf file

Value

plots a dependency tree.

Examples

if (interactive())
    tc_plot_tree(tc_sotu_udpipe, token, lemma, POS)

---

tc_sotu_udpipe A tCorpus with a small sample of sotu paragraphs parsed with udpipe

Description

A tCorpus with a small sample of sotu paragraphs parsed with udpipe

Usage

data(tc_sotu_udpipe)

Format

data.frame
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

```r
tokens_to_tcorpus(
  tokens,
  doc_col = "doc_id",
  token_id_col = "token_id",
  token_col = NULL,
  sentence_col = NULL,
  parent_col = NULL,
  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 occurred (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)
token_col  Optionally, the name of the column that contains the token text. This column will then be renamed to "token" in the tcorpus, which is the default name for many functions (e.g., querying, printing text)
sentence_col  Optionally, the name of the column that indicates the sentences in which tokens occurred. This can be necessary if tokens are not local at the document level (see token_is_local argument), and sentence information can be used in several tcorpus functions.
parent_col  Optionally, the name of the column that contains the id of the parent (if a dependency parser was used). If token_is_local = FALSE, then the token_ids will be transformed, so parent ids need to be changed as well. Default is 'parent', but if this column is not present the parent is ignored.
meta  Optionally, a data.frame with document meta data. Needs to contain a column with the document ids (with the same name)
Alternatively, if there are document meta columns in the tokens data.table, meta_cols can be used to recognize 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 columns 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. !!! if the data has a parent column, make sure to specify parent_col, so that the parent ids are also transformed not used

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 occurred in a matrix.

Description

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

Usage

tokenWindowOccurrence(
  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-occurrence.
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 occurrence of a feature
batch_rows       Used in functions that call this function in batches
drop_empty_terms If TRUE, empty terms (with zero occurrence) 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.

description:  
Show top features

Usage

top_features(
  tc,  
  feature,  
  n = 10,  
  group_by = NULL,  
  group_by_meta = NULL,  
  rank_by = c("freq", "chi2"),  
  dropNA = T,  
  return_long = F  
)

Show top features
transform_rsyntax

Arguments

tc a tCorpus
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.
rank_by The method for ranking the terms. Currently supports frequency (default) and
the 'Chi2' value for the relative frequency of a term in a topic compared to the
overall corpus. If return_long is used, the Chi2 score is also returned, but note
that there are negative Chi2 scores. This is used to indicate that the relative
frequency of a feature in a group was lower than the relative frequency in the
corpus (i.e. under-represented).
dropNA if TRUE, drop NA features
return_long if TRUE, results will be returned in a long format that contains more informa-
tion.

Value
a data.frame

Examples

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

top_features(tc, 'lemma')
top_features(tc, 'lemma', group_by = 'NER', group_by_meta='doc_id')

transform_rsyntax Apply rsyntax transformations

Description

This is an experimental function for applying rsyntax transformations directly on a tcorpus, to cre-
ate a new tcorpus with the transformed tokens. The argument f should be self defined function
that wraps rsyntax transformations. Or more generally, a function that takes a tokens data.frame
(or data.table) as input, and returns a tokens data.frame (or data.table). For examples, see corpus-
tools::ud_relcl, or corpustools::udpipe_simplify for a function that wraps multiple transformations.

Usage

transform_rsyntax(tc, f, ...)
Arguments

- tc: a tCorpus
- f: functions that perform rsyntax tree transformations
- ... arguments passed to f

Value

a tCorpus after applying the transformations

Examples

```r
if (interactive()) {
  tc = tc_sotu_udpipe$copy()
  tc2 = transform_rsyntax(tc, udpipe_simplify)
  browse_texts(tc2)
  rsyntax::plot_tree(tc$tokens, token, lemma, POS, sentence_i=20)
  rsyntax::plot_tree(tc2$tokens, token, lemma, POS, sentence_i=20)
}
```

udpipe_clause_tqueries

*Get a list of tqueries for extracting who did what*

Description

An off-the-shelf list of tqueries for extracting subject-verb clauses. Designed for working with a tCorpus created with `udpipe_tcorpus`.

Usage

`udpipe_clause_tqueries(verbs = NULL, exclude_verbs = verb_lemma("quote"))`

Arguments

- verbs: A character vector for specific verbs to use. By default uses all verbs (except for those specified in exclude_verbs)
- exclude_verbs: A character vector for specific verbs NOT to use. By default uses the verbs that indicate speech (that are used for extracting who said what, in `udpipe_quote_tqueries`)

Examples

```r
udpipe_clause_tqueries()
```
udpipe_quote_tqueries  Get a list of tqueries for extracting quotes

Description
An off-the-shelf list of tqueries for extracting quotes. Designed for working with a tCorpus created with \texttt{udpipe_tcorpus}.

Usage
\begin{verbatim}
udpipe_quote_tqueries(say_verbs = verb_lemma("quote"))
\end{verbatim}

Arguments
say_verbs A character vector of verb lemma that indicate speech (e.g., say, state). A default list is included in \texttt{verb_lemma('quote')}, but certain lemma might be more accurate/appropriate depending on the corpus.

Examples
\begin{verbatim}
udpipe_quote_tqueries()
\end{verbatim}

udpipe_simplify  Simplify tokenIndex created with the udpipe parser

Description
This is an off-the-shelf implementation of several rsyntax transformation for simplifying text.

Usage
\begin{verbatim}
udpipe_simplify(
  tokens,
  split_conj = T,
  rm_punct = F,
  new_sentences = F,
  rm_mark = F
)
\end{verbatim}

Arguments
tokens A tokenIndex, based on output from the ud parser.
split_conj If TRUE, split conjunctions into separate sentences
rm_punct If TRUE, remove punctuation afterwards
new_sentences If TRUE, assign new sentence and token_id after splitting
rm_mark If TRUE, remove children with a mark relation if this is used in the simplification.
udpipe_spanquote_tqueries

Get a list of tqueries for finding candidates for span quotes.

Description
Quote extraction with tqueries is limited to quotes within sentences. When (verbatim) quotes span multiple sentences (which we call span quotes here), they are often indicated with quotation marks. While it is relatively easy to identify these quotes, it is less straightforward to identify the sources of these quotes. A good approach is to first apply tqueries for finding quotes within sentences, because a source mentioned just before (we use 2 sentences) a span quote is often also the source of this span quote. For cases where there is no previous source, we can apply simple queries for finding source candidates. That’s what the tqueries created with the current function are for.

Usage
udpipe_spanquote_tqueries(say_verbs = verb_lemma("quote"))

Arguments

say_verbs A character vector of verb lemma that indicate speech (e.g., say, state). A default list is included in verb_lemma(’quote’), but certain lemma might be more accurate/appropriate depending on the corpus.

Details
This procedure is supported in rsyntax with the add_span_quotes function. In corpustools this function is implemented within the udpipe_quotes method. The current function provides the default tqueries for the span quotes.

Examples
udpipe_spanquote_tqueries()
Description

This is simply shorthand for using create_tcorpus with the udpipe_ arguments and certain specific settings. This is the way to create a tCorpus if you want to use the syntax analysis functionalities.

Usage

udpipe_tcorpus(x, ...)

## S3 method for class 'character'
udpipe_tcorpus(
  x,
  model = "english-ewt",
  doc_id = 1:length(x),
  meta = NULL,
  max_sentences = NULL,
  model_path = getwd(),
  cache = 3,
  cores = NULL,
  batchsize = 50,
  use_parser = T,
  start_end = F,
  verbose = T,
  ...
)

## S3 method for class 'data.frame'
udpipe_tcorpus(
  x,
  model = "english-ewt",
  text_columns = "text",
  doc_column = "doc_id",
  max_sentences = NULL,
  model_path = getwd(),
  cache = 3,
  cores = 1,
  batchsize = 50,
  use_parser = T,
  start_end = F,
  verbose = T,
  ...
)

## S3 method for class 'factor'
udpipe_tcorpus(x, ...)

## S3 method for class 'corpus'
udpipe_tcorpus(x, ...)

**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.
- **...**: Arguments passed to create_tcorpus.character
- **model**: The name of a Universal Dependencies language model (e.g., "english-ewt", "dutch-alpino"), to use the udpipe package (udpipe_annotate). If you don’t know the model name, just type the language and you’ll get a suggestion. Otherwise, use show_udpipe_models to get an overview of the available models. For more information about udpipe and performance benchmarks of the UD models, see the GitHub page of the udpipe package.
- **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
- **max_sentences**: An integer. Limits the number of sentences per document to the specified number.
- **model_path**: If udpipe_model is used, this path will be used to look for the model, and if the model doesn’t yet exist it will be downloaded to this location. Defaults to working directory
- **cache**: The number of persistent caches to keep for inputs of udpipe. The caches store tokens in batches. This way, if a lot of data has to be parsed, or if R crashes, udpipe can continue from the latest batch instead of start over. The caches are stored in the corpustools_data folder (in udpipe_model_path). Only the most recent [udpipe_caches] caches will be stored.
- **cores**: If udpipe_model is used, this sets the number of parallel cores. If not specified, will use the same number of cores as used by data.table (or limited to OMP_THREAD_LIMIT)
- **batchsize**: In order to report progress and cache results, texts are parsed with udpipe in batches of 50. The price is that there will be some overhead for each batch, so for very large jobs it can be faster to increase the batchsize. If the number of texts divided by the number of parallel cores is lower than the batchsize, the texts are evenly distributed over cores.
- **use_parser**: If TRUE, use dependency parser (only if udpipe_model is used)
- **start_end**: If TRUE, include start and end positions of tokens
- **verbose**: If TRUE, report progress. Only if x is large enough to require multiple sequential batches
- **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

```r
## ... if (interactive()) {
  tc = udpipe_tcorpus(c('Text one first sentence. Text one second sentence', 'Text two'),
                   model = 'english-ewt')
  tc$tokens
} if (interactive()) {
  tc = udpipe_tcorpus(sotu_texts[1:5,], doc_column='id', model = 'english-ewt')
  tc$tokens
}
## 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'))
if (interactive()) {
  tc = udpipe_tcorpus(text, 'english-ewt')
  tc$tokens
}
# library(quanteda)
# udpipe_tcorpus(data_corpus_inaugural, 'english-ewt')
```

---

**untokenize**  
*Reconstruct original texts*

**Description**

If the tCorpus was created with remember_spaces = T, you can rebuild the original texts.

**Usage**

`untokenize(tc)`

**Arguments**

- `tc`: A tCorpus, created with `create_tcorpus`, with remember_spaces = TRUE

**Value**

A data.table with the text fields and meta fields as columns.

**Examples**

```r
tc = create_tcorpus(sotu_texts, doc_column='id')
untokenize(tc)
```
Index

* datasets
  - corenlp_tokens, 19
  - sotu_texts, 64
  - stopwords_list, 65
  - tc_sotu_udpipe, 99
  - Co-occurrence networks, 69
  - code_dictionary
    - (tCorpus$code_dictionary), 70
  - code_features (tCorpus$code_features), 72
  - compare_corpus, 15, 43
  - compare_corpus(), 94
  - compare_documents, 16
  - compare_documents(), 96
  - compare_subset, 18, 43
  - compare_subset(), 94
  - context (tCorpus$context), 73
  - corenlp_tokens, 19
  - Corpus comparison, 69
  - count_tcorpus, 19
  - Create a tCorpus, 69
  - create_tcorpus, 20, 29, 76, 109
  - create_tcorpus(), 94
  - deduplicate (tCorpus$deduplicate), 73
  - delete_columns
    - (tCorpus$delete_columns), 75
  - delete_meta_columns
    - (tCorpus$delete_columns), 75
  - docfreq_filter, 24
  - Document similarity, 69
  - dtm_compare, 25
  - dtm_wordcloud, 26
  - ego_semnet, 27
  - ego_semnet(), 98
  - export_span_annotations, 28
  - feats_to_columns
    - (tCorpus$feats_to_columns), 76
  - feature_associations, 29, 42
feature_associations(), 97
feature_stats, 31
feature_stats(), 96
feature_subset
(tCorpus$feature_subset), 76
Features, 69
fold_rsyntax, 32
freq_filter, 33
get(tCorpus$get), 78
get_dfm(get_dtm), 33
get_dfm(), 95
get_dtm, 33
get_dtm(), 94
get_global_i, 36
get_kwic, 36
get_meta(tCorpus$get), 78
get_stopwords, 38
kwic(), 97
laplace, 39
lda_fit(tCorpus$lda_fit), 80
Manage tCorpus data, 69
melt_quanteda_dict, 39, 55, 71, 84
merge(tCorpus$merge), 81
merge_meta(tCorpus$merge), 81
merge_tcorpora, 40, 95
modified by reference, 69
plot.contextHits, 41
plot.featureAssociations, 42
plot.featureHits, 42
plot.vocabularyComparison, 43
plot_semet, 44
plot_semet(), 98
plot_tree, 98, 99
plot_words, 46
preprocess, 23
preprocess(tCorpus$preprocess), 82
preprocess_tokens, 47
print.contextHits, 49
print.featureHits, 49
print.tCorpus, 50
refresh_tcorpus, 50
replace_dictionary
(tCorpus$replace_dictionary), 83
require_package, 51
search_contexts, 18, 20, 41, 49, 52, 66, 67, 90, 95
search_contexts(), 97
search_dictionary, 20, 54, 83
search_features, 11, 20, 30, 37, 43, 50, 56, 68, 72, 85
search_features(), 97
search_recode(tCorpus$search_recode), 85
semnet, 60
semnet(), 98
semnet_window, 61
semnet_window(), 98
set(tCorpus$set), 86
set_levels(tCorpus$set_levels), 87
set_meta(tCorpus$set), 86
set_meta_levels(tCorpus$set_levels), 87
set_meta_name(tCorpus$set_name), 88
set_name(tCorpus$set_name), 88
set_network_attributes, 62
sgt, 63
show_udpipe_models, 22, 64, 108
sotu_texts, 64
stopwords_list, 65
subset, 66, 90, 95
subset(tCorpus$subset), 88
subset(), 95
subset.data.table, 89
subset.tCorpus, 65
subset_meta(tCorpus$subset), 88
subset_query, 38, 66
subset_query(), 95
summary.contextHits, 67
summary.featureHits, 67
summary.tCorpus, 68
tc_plot_tree, 98
tc_sotu_udpipe, 99
tCorpus, 15, 16, 18, 20, 30, 34, 52, 56, 66, 68
tcorpus(tCorpus), 68
tCorpus$annotate_rsyntax, 69
tCorpus$code_dictionary, 70
tCorpus$code_features, 72
tCorpus$context, 73
tCorpus$deduplicate, 73
tCorpus$delete_columns, 75
tCorpus$delete_meta_columns
    (tCorpus$delete_columns), 75
Corpus$feats_to_columns, 76
Corpus$feature_subset, 76, 89
Corpus$fold_rsyntax, 77
tCorpus$get, 78
tCorpus$get_meta (tCorpus$get), 78
tCorpus$lda_fit, 80
tCorpus$merge, 81
tCorpus$preprocess, 82
tCorpus$replace_dictionary, 83
tCorpus$search_recode, 85
tCorpus$set, 86
tCorpus$set_levels, 87
tCorpus$set_meta (tCorpus$set), 86
tCorpus$set_meta_levels
    (tCorpus$set_levels), 87
tCorpus$set_meta_name
    (tCorpus$set_name), 88
tCorpus$set_name, 88
tCorpus$subset, 77, 79, 88, 95
tCorpus$subset_meta (tCorpus$subset), 88
tCorpus$subset_query, 90
tCorpus$udpipe_clauses, 91
tCorpus$udpipe_coref, 92
tCorpus$udpipe_quotes, 92
tCorpus_compare, 94
tCorpus_create, 94
tCorpus_data, 94
tCorpus_docsim, 95
tCorpus_features, 96
tCorpus_modify_by_reference, 73, 96
tCorpus_querying, 97
tCorpus_semnet, 98
tCorpus_topmod, 98
tokens_to_tcorpus, 100
tokens_to_tcorpus(), 94
tokenWindowOccurence, 101
top_features, 102
top_features(), 96
Topic modeling, 69
tquery, 70
transform_rsyntax, 103

udpipeannotate, 22, 108
udpipe_clause_tqueries, 91, 104
udpipe_clauses
    (tCorpus$udpipe_clauses), 91
udpipe_coref (tCorpus$udpipe_coref), 92
udpipe_download_model, 64
udpipe_quote_tqueries, 92, 93, 104, 105
udpipe_quotes, 106
udpipe_quotes (tCorpus$udpipe_quotes), 92
udpipe_simplify, 105
udpipe_spanquote_tqueries, 92, 93, 106
udpipe_tcorpus, 91, 92, 104, 105, 107
untokenize, 109
Using search strings, 69