Package 'termco'

April 14, 2017

Title Counts of Terms and Substrings

Version 0.5.2

Maintainer Tyler Rinker <tyler.rinker@gmail.com>

Description A small suite of functions used to count terms and substrings in strings.

Depends R (>= 3.2.1)

Imports ca, clipr, data.table, dplyr (>= 0.4.3), ggplot2, ggnetwork, gofastr, graphics, grDevices, grid, gridExtra, igraph, methods, quanteda, slam, SnowballC, stats, stringi, textshape, tidyr, tm, utils, wordcloud

Suggests intergraph, knitr, lexicon, rmarkdown, qdapRegex, rgl, testthat

Date 2017-04-14

License MIT + file LICENSE

LazyData TRUE

Roxygen list(wrap = FALSE)

RoxygenNote 6.0.1

ByteCompile true

BugReports https://github.com/trinker/termco/issues?state=open

URL https://github.com/trinker/termco

Author Tyler Rinker [aut, cre], Steven Simpson [ctb]

RemoteType local

RemoteUrl C:\{}Users\{}Tyler\{}GitHub\{}termco

RemoteSha NA

RemoteBranch master

RemoteUsername trinker

RemoteRepo termco

R topics documented:

as_count . . . 3
as_terms . . . 4
as_term_list . . . 4
classification_project . . . 5
classify . . . 6
collapse_tags . . . 7
colo . . . 8
combine_counts . . . 9
coverage . . . 10
evaluate . . . 12
frequent_terms . . . 13
frequent_terms_co_occurrence . . . 15
get_text . . . 17
hierarchical_coverage_regex . . . 18
hierarchical_coverage_term . . . 19
important_terms . . . 20
markers . . . 21
ngram_collocations . . . 21
plot.classify . . . 23
plot.frequent_terms . . . 23
plot.hierarchical_coverage_regex . . . 24
plot.hierarchical_coverage_term . . . 24
plot.important_terms . . . 25
plot.ngram_collocations . . . 25
plot.tag_co_occurrence . . . 26
plot.term_count . . . 27
plot.term_loc . . . 27
plot.token_count . . . 28
plot.validate_model . . . 29
plot_ca . . . 29
plot_counts . . . 30
plot_cum_percent . . . 31
plot_freq . . . 32
presidential_debates_2012 . . . 33
print.as_terms . . . 34
print.combine_counts . . . 34
print.coverage . . . 34
print.evaluate . . . 35
print.frequent_terms . . . 35
print.frequent_terms_co_occurrence . . . 36
print.hierarchical_coverage . . . 36
print.hierarchical_coverage_regex . . . 37
print.important_terms . . . 37
print.probe_list . . . 38
print.search_term . . . 38
print.split_data . . . 38
print.summary.validate_model . . . 39
print.summary_hierarchical_coverage_regex . . . 39
print.term_count . . . 40
print.token_count . . . 40

print.validate_model 41
probe_colo_list 41
probe_colo_plot_list 42
probe_list 44
sam_i_am 45
search_term 45
search_term_collocations 46
split_data 47
summary.hierarchical_coverage_regex 48
summary.validate_model 48
tag_co_occurrence 49
termco 51
term_before 51
term_cols 52
term_count 53
token_count 55
uncovered 57
unnest_term_list 58
update_names 59
validated 60
validate_model 60
weight 62

Index **63**

`as_count` *Set Pretty/Count Printing of a term_count Object*

Description

`as_count` - Set the pretty printing of a `term_count` object. Either print pretty as a combination of count and percent/proportion or as just counts.

`pretty<-` - Set the pretty/count printing of a `term_count` object.

Usage

```
as_count(x, value = FALSE)

as_count(x) <- value
```

Arguments

`x`	A `term_count` object.
`value`	logical. If `TRUE` the object will attempt to be printed pretty.

Details

Note that pretty printing can be turned off globally by setting `options(termco_pretty = FALSE)`.

Value

Returns a `term_count` with the `pretty` `attributes` set.

See Also

as_count

Examples

```
out <- as_count(markers, FALSE)
out
as_count(out) <- TRUE
out
```

---

as_terms *Convert a Count Matrix to List of Term Vectors*

Description

Convert a count matrix to a named list of term vectors.

Usage

```
as_terms(x, names = NULL, ...)
```

Arguments

x	A data.frame/matrix of counts.
names	A character vector of names to assign to the list.
...	ignored.

Value

Returns a list of term vectors.

Examples

```
data(markers)
as_terms(markers)
```

---

as_term_list *Coerce to Named List*

Description

Convenience function to convert a data forms of terms into a named list. For vectors, names are the same as the terms.

Usage

```
as_term_list(x, add.boundary = FALSE, ...)
```

Arguments

x	A vector of strings or a **quanteda** dictionary.
add.boundary	logical. If TRUE a word boundary is place at the beginning and end of the strings.
...	ignored.

Value

Returns a named list.

Examples

```
as_term_list(state.name)

## Not run:
if (!require("pacman")) install.packages("pacman")
pacman::p_load(tidyverse)


x <- presidential_debates_2012[["dialogue"]]

bigrams <- ngram_collocations(x, n=10) %>%
    transmute(bigram = paste(term1, term2)) %>%
    unlist() %>%
    as_term_list()


presidential_debates_2012 %>%
    with(term_count(dialogue, person, bigrams))

## dictionary from quanteda
require(quanteda)
mfdict <- dictionary(
    file = "http://ow.ly/VMRkL",
    format = "LIWC"
)

as_term_list(mfdict, TRUE)

## End(Not run)
```

---

classification_project

Classification Project Template

Description

Create a template directory of subdirectories and files for a classification project.

Usage

```
classification_project(path = "new", open = is.global(2))

is.global(n = 1)
```

Arguments

path	The path (including project name) for the project.
open	logical. If TRUE the project will be opened in RStudio. The default is to test if new_project is being used in the global environment, if it is then the project directory will be opened.
n	The number of generations to go back. If used as a function argument n should be set to 2.

Examples

```
## Not run:
classification_project()

## End(Not run)
```

---

classify *Classify Rows of a Count Matrix*

Description

Use a count matrix to classify the rows, based on the frequencies in the cells.

Usage

```
classify(x, n = 1, ties.method = "probability", seed = NULL, ...)
```

Arguments

x	A term_count or count matrix/data.frame.
n	The number of classifications per row to return.
ties.method	Either c("probability", "random", "first", "last") for specifying how ties are handled; "probability" by default. This utilizes the probability distributions from all tags (regardless of strength/counts of tags) to randomly sample with probabilties to break ties. Note that this can lead to different results each time classify is run. Use seed to make results reproducible. The other methods use max.col for tie breaking. See max.col for a description of those arguments.
seed	A seed to use in the sample to make the results reproducible.
...	ignored.

Value

Returns a single vector or list of ordered vectors of predicted classifications; order by term frequency. Ties default to random order.

See Also

max.col

Examples

```
## Not run:
library(dplyr)
data(presidential_debates_2012)

discoure_markers <- list(
    response_cries = c("\\boh", "\\bah", "\\baha", "\\bouch", "yuk"),
    back_channels = c("uh[- ]huh", "uhuh", "yeah"),
    summons = "hey",
    justification = "because"
)

presidential_debates_2012 %>%
    with(., term_count(dialogue, TRUE, discoure_markers)) %>%
    classify()

presidential_debates_2012 %>%
    with(., term_count(dialogue, TRUE, discoure_markers)) %>%
    classify() %>%
    plot()

presidential_debates_2012 %>%
    with(., term_count(dialogue, TRUE, discoure_markers)) %>%
    classify() %>%
    plot(rm.na=FALSE)

presidential_debates_2012 %>%
    with(., term_count(dialogue, TRUE, discoure_markers)) %>%
    classify(n = 2)

presidential_debates_2012 %>%
    with(., term_count(dialogue, TRUE, discoure_markers)) %>%
    {.[!uncovered(.), -c(1:2)]} %>%
    classify()

## End(Not run)
```

`collapse_tags` *Collapse* `term_count` *Tags*

Description

Collapse (sum) tags/columns of a `term_count` object or remove columns without changing termco class.

Usage

```
collapse_tags(x, mapping, ...)
```

Arguments

`x`	A `term_count` object.

mapping	A list of named vectors where the vector names are the collapsed column names and the vectors are the names of the columns to collapse. Setting a column name to NULL deletes these columns from the output.
...	ignored.

Value

Returns a term_count object.

Examples

```
mapping <- list(
    babbling = c('response_cries', 'back_channels'), #combines these columns
    NULL = 'justification'                           #remove this column(s)
)

data(markers); markers
collapse_tags(markers, mapping)
```

---

colo *Make Regex to Locate Strings Containing Co-ocuring Substrings*

Description

Make a regex to locate strings that contain >= 2 substrings with optional negation.

Usage

```
colo(..., not = NULL, copy2clip = getOption("termco.copy2clip"))
```

Arguments

not	A substring to exclude from consideration.
copy2clip	logical. If codeTRUE uses write_clip to copy the output to the clipboard. This option is most useful when trying to build a list regular expression model for easy pasting between testing a regex and putting it into the model. This argument can be set globally by setting options(termco.copy2clip = TRUE).
...	Terms that cooccur/collocate

Value

Returns a regular expression. If Windows attempts to copy to clipboard as well.

Examples

```
## Not run:
colo('overall', 'course')
colo('overall', 'course', "eval")
colo('overall', 'course', not="instructor")

search_term(sam_i_am, colo("^i\\b", "like"))
search_term(sam_i_am, colo("^i\\b", "like", "not"))
```

```
search_term(sam_i_am, colo("^i\\b", "like|not"))
search_term(sam_i_am, colo("^i\\b", "like", not="not"))

## End(Not run)
```

---

combine_counts *Combine* term_count *and* token_count *Objects*

Description

Combine term_count and token_count objects.

Usage

```
combine_counts(x, y, mapping = NULL, ...)
```

Arguments

x	A term_count or token_count object.
y	A term_count or token_count object.
mapping	A list of named vectors where the vector names are the collapsed column names and the vectors are the names of the columns to collapse. The default, NULL, combines columns with the same name in both x and y.
...	ignored.

Value

Returns a combine_counts object.

Examples

```
token_list <- list(
    list(
        person = c('sam', 'i')
    ),
    list(
        place = c('here', 'house'),
        thing = c('boat', 'fox', 'rain', 'mouse', 'box', 'eggs', 'ham')
    ),
    list(
        no_like = c('not like'),
        thing = c('train', 'goat')
    )
)

(y <- token_count(sam_i_am, grouping.var = TRUE, token.list = token_list))


term_list <- list(
    list(Is = c("I")),
    list(
        oulds = c("ould"),
        thing = c('egg', 'ham')
```

```
    )
)


(x <-term_count(sam_i_am, grouping.var = TRUE, term_list, ignore.case = FALSE))

combine_counts(x, y)
combine_counts(y, x)

library(lexicon)
library(textshape)
library(dplyr)

token_list <- lexicon::nrc_emotions %>%
    textshape::column_to_rownames() %>%
    t() %>%
    textshape::as_list()


a <- presidential_debates_2012 %>%
     with(token_count(dialogue, list(person, time), token_list))

term_list <- list(
    response_cries = c("\\boh", "\\bah", "\\baha", "\\bouch", "yuk"),
    back_channels = c("uh[- ]huh", "uhuh", "yeah"),
    summons = "hey",
    justification = "because"
)

b <- presidential_debates_2012 %>%
     with(term_count(dialogue, list(person, time), term_list))

combine_counts(a, b)
combine_counts(b, a)


d <- sam_i_am %>%
     term_count(TRUE, token_list[1:2])

e <- sam_i_am %>%
     term_count(TRUE, token_list[[3]])


combine_counts(e, d)
```

---

coverage *Coverage for Various Objects*

Description

coverage - Get coverage of a logical vector, term_count, or search_term object.

coverage.term_count - Extract coverage information from a term_count object including the percentage of rows that sum to zero as well as the location of non-covered rows for easy extraction.

Usage

```
coverage(x, ...)
```

Arguments

x	A logical vector, termc_count, or search_term object.
...	Ignored.

Value

term_count - Returns a proportion of elements covered by the search.

coverage.term_count - Returns a list:

not	A logical vector of all rows not covered (row sums equal zero)
covered	A logical vector of all rows covered (row sums greater than zero)
coverage	The percentage rate of $\frac{covered}{not + covered}$
n_covered	The row sums of the unique terms
total_terms	The row sums of the terms
hierarchical_covered*	A hierarchical list (matching the term.list structure) of logical vectors of all rows covered (row sums greater than zero)
hierarchical_n_covered*	A hierarchical vector (matching the term.list structure) of the row sums of the unique terms
hierarchical_coverage*	A hierarchical vector (matching the term.list structure) of the percentage rate of $\frac{covered}{not + covered}$

**Only applies to* term_count *output that was generated with a hierarchical* term.list

Examples

```
coverage(sample(c(TRUE, FALSE), 1000, TRUE))

data(presidential_debates_2012)

discoure_markers <- list(
    like = c("love", "like"),
    water = c("lake", "ocean", "water"),
    justify = c("because"),
    he = c("\\bhe", "him"),
    we = c("\\bwe", "\\bus", "\\bour")
)

library(dplyr)
(markers2 <- with(presidential_debates_2012,
    term_count(dialogue, TRUE, discoure_markers)
))
```

```
coverage(markers2)

presidential_debates_2012[coverage(markers2)$not, "dialogue"] %>%
   c()
```

`evaluate` *Model Evaluation*

Description

Get accuracy, precision, and recall for multi-class, multi-tag, predictions.

Usage

```
evaluate(x, known)
```

Arguments

`x`	The model classification `list`/`vector` (typically the results of `classify`).
`known`	The known expert coded `list`/`vector` of outcomes.

Value

Returns a list of seven elements:

`N`	The number of elements being assessed
`confusion_matrix`	A list of confusion matrices for each tag
`tag_accuracy`	A tag named vector of accuracies computed from the confusion matrices; (tp + tn)/(tp + tn + fp + fn)
`tag_precision`	A tag named vector of precisions computed from the confusion matrices; tp/(tp + fp)
`tag_recall`	A tag named vector of accuracies computed from the confusion matrices; tp/(tp + fn)
`macro_averaged`	Macro averaged accuracy, precision, and recall; computed accuracy, precision, and recall for each confusion matrix and average
`micro_averaged`	Micro averaged accuracy, precision, and recall; add the confusion amtrices and compute accuracy, precision, and recall

References

https://www.youtube.com/watch?v=OwwdYHWRB5E&index=31&list=PL6397E4B26D00A269

Examples

```
known <- list(1:3, 3, NA, 4:5, 2:4, 5, integer(0))
tagged <- list(1:3, 3, 4, 5:4, c(2, 4:3), 5, integer(0))
evaluate(tagged, known)

## Examples
library(dplyr)
data(presidential_debates_2012)

discoure_markers <- list(
    response_cries = c("\\boh", "\\bah", "\\baha", "\\bouch", "yuk"),
    back_channels = c("uh[- ]huh", "uhuh", "yeah"),
    summons = "hey",
    justification = "because"
)


## Only Single Tag Allowed Per Text Element
mod1 <- presidential_debates_2012 %>%
    with(., term_count(dialogue, TRUE, discoure_markers)) %>%
    classify()

fake_known <- mod1
set.seed(1)
fake_known[sample(1:length(fake_known), 300)] <- "random noise"

evaluate(mod1, fake_known)

## Multiple Tags Allowed
mod2 <- presidential_debates_2012 %>%
    with(., term_count(dialogue, TRUE, discoure_markers)) %>%
    classify(n = 2)

fake_known2 <- mod2
set.seed(30)
fake_known2[sample(1:length(fake_known2), 500)] <- c("random noise", "back_channels")

(myacc <- evaluate(mod2, fake_known2))
myacc$confusion_matrix
myacc$tag_accuracy
```

`frequent_terms` *N Most Frequent Terms*

Description

`frequent_terms` - Find a list of the n most frequent terms.

`all_words` - Find a list of all terms used.

Usage

```
frequent_terms(text.var, n = 20, stopwords = tm::stopwords("en"),
  min.freq = NULL, min.char = 4, max.char = Inf, stem = FALSE,
```

```
  language = "porter", strip = TRUE, strip.regex = "[^a-z' ]",
  alphabetical = FALSE, ...)

all_words(text.var, stopwords = NULL, min.char = 0, ...)
```

Arguments

text.var	A vector of character strings.
n	The number of rows to print. If integer selects the frequency at the nth row and prints all rows >= that value. If proportional (less than 0) the frequency value for the nth% row is selected and prints all rows >= that value.
stopwords	A vector of stopwords to exclude.
min.freq	The minimum frequency to print. Note that this argument overides the n argument.
min.char	The minimum number of characters a word must be (including apostrophes) for inclusion.
max.char	The maximum number of characters a word must be (including apostrophes) for inclusion.
stem	logical. If TRUE the wordStem is used with language = "porter" as the default. Note that stopwords will be stemmed as well.
language	The stem language to use (see wordStem).
strip	logical. If TRUE all values that are not alpha, apostrophe, or spaces are stripped. This regex can be changed via the strip.regex argument.
strip.regex	A regular expression used for stripping undesired characters.
alphabetical	logical. Should rows be arranged alphabetically by term or frequency.
...	ignored.

Value

Returns a data.frame of terms and frequencies.

Examples

```
## Not run:
x <- presidential_debates_2012[["dialogue"]]

frequent_terms(x)
frequent_terms(x, min.char = 1)
frequent_terms(x, n = 50)
frequent_terms(x, n = .02)
frequent_terms(x, stem = TRUE)
frequent_terms(x, n = 50, stopwords = c(tm::stopwords("en"), "said", "well"))

plot(frequent_terms(x))
plot(frequent_terms(x, n = .02))
plot(frequent_terms(x, n = 40))
plot(frequent_terms(x, n = 40), as.cloud = TRUE)

## Note `n` can be used in print to change how many rows are returned.
## This output can be reassigned when wrapped in print.  This is useful
## reduce computational time on larger data sets.
```

```
y <- frequent_terms(x, n=10)
nrow(y)
z <- print(frequent_terms(x, n=100))
nrow(z)

## Cumulative Percent Plot
plot_cum_percent(frequent_terms(presidential_debates_2012[["dialogue"]]))

## End(Not run)
```

`frequent_terms_co_occurrence` *Plot Co-Occurrence of Frequent Terms*

Description

Generate a `tag_co_occurrence` object from frquent terms.

Usage

```
frequent_terms_co_occurrence(x, bound = TRUE, ...)
```

Arguments

x	A vector of character strings.
bound	ligical. If `TRUE` each side of the frequent term is wrapped with a word boundary before performing thre regex search. Otherwise, the search is fuzzy matched.
...	Other arguments passed to `frequent_terms`.

Value

Returns a `tag_co_occurrence` object from frequent terms.

Examples

```
## Not run:
frequent_terms_co_occurrence(presidential_debates_2012[["dialogue"]])
frequent_terms_co_occurrence(presidential_debates_2012[["dialogue"]], bound = FALSE)

x <- frequent_terms_co_occurrence(presidential_debates_2012[["dialogue"]], n=50)
x
plot(x, min.edge.cutoff = .1, node.color = "gold")
plot(x, min.edge.cutoff = .075, node.color = "#1CDB4F")

## Load Required Add-on Packages
if (!require("pacman")) install.packages("pacman")
pacman::p_load(igraph, qrage)
pacman::p_load_gh("mattflor/chorddiag", "trinker/textshape")

## Matrix Manipulation Function
remove_diags <- function(mat, rm.lower = FALSE, order = TRUE, ...) {
    diag(mat) <- 0
    if (isTRUE(rm.lower)) mat[lower.tri(mat)] <- 0
```

```
    if (order) {
        ord <- order(rowSums(mat))
        mat <- mat[ord, ord]
    }
    mat
}

##--------------
## Chord Diagram
##--------------
chorddiag::chorddiag(
    remove_diags(x[["adjacency"]]),
    margin = 150,
    showTicks =FALSE,
    groupnamePadding = 5,
    groupThickness = .05,
    chordedgeColor = NA
)

add_diags <- function(x, y, ...){
    diag(x) <- y
    x
}

order_tags <- function(x, ...){
    ord <- order(rowSums(x))
    x[ord, ord]
}

remove_lower <- function(x, ...){
    x[lower.tri(x)] <- 0
    x
}


chorddiag::chorddiag(
    add_diags(x[["adjacency"]],x[["node_size"]])  %>% order_tags() %>% remove_lower(),
    margin = 150,
    showTicks =FALSE,
    groupnamePadding = 5,
    groupThickness = .05,
    chordedgeColor = NA
)

chorddiag::chorddiag(
    x[["adjacency"]] %>% order_tags() %>% remove_lower(),
    margin = 150,
    showTicks =FALSE,
    groupnamePadding = 5,
    groupThickness = .05,
    chordedgeColor = NA
)

##--------------
## Network Graph
##--------------
graph <- igraph::graph.adjacency(
```

```
    remove_diags(x[["adjacency"]], order=FALSE),
    weighted = TRUE
)

linkdf <- stats::setNames(get.data.frame(graph), c("source", "target", "value"))

qrage::qrage(
    links = linkdf,
    nodeValue = textshape::tidy_vector(x[['node_size']]),
    cut = 0.1
)

## End(Not run)
```

`get_text` *Get a Text Stored in Various Objects*

Description

Extract the text supplied to the `term_count` object.

Usage

```
get_text(x, ...)

## S3 method for class 'term_count'
get_text(x, ...)
```

Arguments

`x`	A `term_count` object.
`...`	`term_count` tags.

Value

Returns a vector or list of text strings.

Examples

```
library(dplyr)

discoure_markers <- list(
    response_cries = c("\\boh\\b", "\\bah\\b", "\\baha", "\\bouch", "yuk"),
    back_channels = c("uh[- ]huh", "uhuh", "yeah"),
    summons = "\\bhey",
    justification = "because"
)

model <- presidential_debates_2012 %>%
    with(term_count(dialogue, grouping.var = TRUE, discoure_markers))

get_text(model, 'summons')
get_text(model, 'response_cries')
get_text(model, c('summons', 'response_cries'))
```

`hierarchical_coverage_regex` *Hierarchical Coverage of Regexes*

Description

The unique coverage of a text vector by a regex after partitioning out the elements matched by previous regexes.

Usage

```
hierarchical_coverage_regex(text.var, term.list, ignore.case = TRUE,
  sort = FALSE, verbose = TRUE, ...)
```

Arguments

`text.var`	A text vector (vector of strings).
`term.list`	A list of named character vectors to match against `x`.
`ignore.case`	logical. Should case be ignored in matching the `terms` against `x`?
`sort`	logical. If `TRUE` the output is sorted by highest unique gain. If `FALSE` order of term input is retained.
`verbose`	If `TRUE` each iteration of the `for` loop prints `i of n`.
`...`	ignored.

Value

Returns a `data.frame` with 7 columns:

step the order in which the regex was searched for

name the human readable name of the bound regex group

unique_prop the unique prop coverage of the regex

unique_n the unique n coverage of the regex

cum_prop the cumulative prop coverage of the regex

cum_n the cumulative n coverage of the regex

regex the bound (|) regex that corresponds to `name`

See Also

Other hierarchical_coverage functions: `hierarchical_coverage_term`

Examples

```
regs <- setNames(
    list(c('(?i)sam', "(?i)\\bam"), '^I', '(?i)(do|will) not', '(?i)(do|will)'),
    c('am', 'I', "won't")
)
(out <- hierarchical_coverage_regex(sam_i_am, regs, ignore.case=FALSE))
summary(out)
plot(out)
```

```
plot(out, mark.one = TRUE)

# Use unnamed vectors for `term.list` too
hierarchical_coverage_regex(sam_i_am, unlist(regs, use.names = FALSE), ignore.case=FALSE)
```

`hierarchical_coverage_term` *Hierarchical Coverage of Terms*

Description

The unique coverage of a text vector by a term after partitioning out the elements matched by previous terms.

Usage

```
hierarchical_coverage_term(text.var, terms, bound = TRUE,
  ignore.case = TRUE, sort = FALSE, ...)
```

Arguments

`text.var`	A text vector (vector of strings).
`terms`	A vector of regular expressions to match against `x`.
`bound`	logical. If `TRUE` the terms are bound with boundary markers to ensure `"read"` matches `"read"` but not `"ready"`).
`ignore.case`	logical. Should case be ignored in matching the `terms` against `x`?
`sort`	logical. If `TRUE` the output is sorted by highest unique gain. If `FALSE` order of term input is retained.
`...`	ignored.

Value

Returns a `data.frame` with 3 columns:

terms the search term

unique the unique coverage of the term

cumulative the cumulative coverage of the term

Author(s)

Steve T. Simpson and Tyler Rinker <tyler.rinker@gmail.com>.

See Also

Other hierarchical_coverage functions: `hierarchical_coverage_regex`

Examples

```
x <- presidential_debates_2012[["dialogue"]]
terms <- frequent_terms(x)[[1]]
(out <- hierarchical_coverage_term(x, terms))
plot(out)

(out2 <- hierarchical_coverage_term(x, frequent_terms(x, 30)[[1]]))
plot(out2, use.terms = TRUE)
plot(out2, use.terms = TRUE, mark.one = TRUE)
```

`important_terms` *Top Min-Max Scaled TF-IDF terms*

Description

View the top n min-max scaled tf-idf weighted terms in a text.

Usage

```
important_terms(text.var, n = 20, stopwords = tm::stopwords("en"),
  stem = FALSE, language = "porter", strip = TRUE,
  strip.regex = "[^A-Za-z' ]", ...)
```

Arguments

`text.var`	A vector of character strings.
`n`	The number of rows to print. If integer selects the frequency at the nth row and prints all rows >= that value. If proportional (less than 0) the frequency value for the nth% row is selected and prints all rows >= that value.
`stopwords`	A vector of stopwords to exclude.
`stem`	logical. If `TRUE` the `wordStem` is used with `language = "porter"` as the default. Note that stopwords will be stemmed as well.
`language`	The stem language to use (see `wordStem`).
`strip`	logical. If `TRUE` all values that are not alpha, apostrophe, or spaces are stripped. This regex can be changed via the `strip.regex` argument.
`strip.regex`	A regular expression used for stripping undesired characters.
`...`	`remove_stopwords`

Value

Returns a `data.frame` of terms and min-max scaled tf-idf weights.

Examples

```
## Not run:
x <- presidential_debates_2012[["dialogue"]]

frequent_terms(x)
important_terms(x)
important_terms(x, n=899)
```

```
important_terms(x, n=.1)
important_terms(x, min.char = 7)
important_terms(x, min.char = 6, stem=TRUE)

plot(important_terms(x))
plot(important_terms(x, n = .02))
plot(important_terms(x, n = 40))
plot(important_terms(x, n = 100), as.cloud = TRUE)

## End(Not run)
```

`markers` *Discourse Marker Search of Presidential Debates*

Description

A `term_count` dataset containing discourse markers from the 2012 presidential debates.

Usage

```
data(markers)
```

Format

A data frame with 10 rows and 7 variables

Details

- person. The speaker
- time. Variable indicating which of the three debates the dialogue is from
- n.words. The number of words
- response_cries. The number of response cries: `c("oh", "ah", "aha", "ouch", "yuk")`
- back_channels. The number of back channels: `c("uh[- ]huh", "uhuh", "yeah")`
- summons. The number of summons: `"hey"`
- justification. The number of justification: `"because"`

`ngram_collocations` *Ngram Collocations*

Description

Find a important ngram (2-3) collocations. Wraps `collocations` to provide stopword, min/max characters, and stemming with a generic `plot` function.

Usage

```
ngram_collocations(text.var, n = 20, gram.length = 2,
  stopwords = tm::stopwords("en"), min.char = 4, max.char = Inf,
  order.by = "frequency", stem = FALSE, language = "porter", ...)
```

Arguments

text.var	A vector of character strings.
n	The number of rows to include.
gram.length	The length of ngram to generate (2-3).
stopwords	A vector of stopwords to exclude.
min.char	The minimum number of characters a word must be (including apostrophes) for inclusion.
max.char	The maximum number of characters a word must be (including apostrophes) for inclusion.
order.by	The name of the measure column to order by: "frequency", "G2", "X2", "pmi", "dice".
stem	logical. If TRUE the wordStem is used with language = "porter" as the default. Note that stopwords will be stemmed as well.
language	The stem language to use (see wordStem).
...	Other arguments passed to collocations.

Value

Retuns a data.frame of terms and frequencies.

See Also

collocations

Examples

```
## Not run:
x <- presidential_debates_2012[["dialogue"]]

ngram_collocations(x)
ngram_collocations(x, n = 50)
ngram_collocations(x, stopwords = c(tm::stopwords("en"), "american", "governor"))
ngram_collocations(x, gram.length = 3)
ngram_collocations(x, gram.length = 3, stem = TRUE)
ngram_collocations(x, order.by = "dice")

plot(ngram_collocations(x))
plot(ngram_collocations(x, n = 40))
plot(ngram_collocations(x, order.by = "dice"))
plot(ngram_collocations(x, gram.length = 3))

## End(Not run)
```

plot.classify *Plots a plot.classify Object*

Description

Plots a plot.classify object

Usage

```
## S3 method for class 'classify'
plot(x, rm.na = TRUE, ...)
```

Arguments

x	A classify object.
rm.na	logical. If TRUE unclassified NA values are not displayed.
...	Other arguments passed to plot_counts.

plot.frequent_terms *Plots a frequent_terms Object*

Description

Plots a frequent_terms object.

Usage

```
## S3 method for class 'frequent_terms'
plot(x, n, as.cloud = FALSE, random.order = FALSE,
  rot.per = 0, ...)
```

Arguments

x	The frequent_terms object.
n	The number of rows to plot. If integer selects the frequency at the nth row and plots all rows >= that value. If proportional (less than 0) the frequency value for the nth% row is selected and plots all rows >= that value.
as.cloud	logical. If TRUE a wordcloud will be plotted rather than a bar plot.
random.order	logical. Should the words be place randomly around the cloud or if FALSE the more frequent words are in the center of the cloud.
rot.per	The precentage of rotated words.
...	Other arguments passed to wordcloud.

plot.hierarchical_coverage_regex

Plots a hierarchical_coverage_regex Object

Description

Plots a hierarchical_coverage_regex object

Usage

```
## S3 method for class 'hierarchical_coverage_regex'
plot(x, use.names = nrow(x) <= 30,
  mark.one = FALSE, sort = FALSE, ...)
```

Arguments

x	A hierarchical_coverage_regex object.
use.names	logical. If TRUE terms are plotted on the x axis. If FALSE word numbers are. Te default is to plot terms if they are equal to or less than 30 in length.
mark.one	logical. If TRUE a purple horizontal line is added at 100% and the y axis is extended as well.
sort	logical. If TRUE the regex terms are sorted by highest unique gain.
...	ignored.

plot.hierarchical_coverage_term

Plots a hierarchical_coverage_term Object

Description

Plots a hierarchical_coverage_term object

Usage

```
## S3 method for class 'hierarchical_coverage_term'
plot(x, use.terms = nrow(x) <= 30,
  mark.one = FALSE, sort = FALSE, ...)
```

Arguments

x	A hierarchical_coverage_term object.
use.terms	logical. If TRUE terms are plotted on the x axis. If FALSE word numbers are. Te default is to plot terms if they are equal to or less than 30 in length.
mark.one	logical. If TRUE a purple horizontal line is added at 100% and the y axis is extended as well.
sort	logical. If TRUE the terms are sorted by highest unique gain.
...	ignored.

plot.important_terms *Plots a important_terms Object*

Description

Plots a important_terms object.

Usage

```
## S3 method for class 'important_terms'
plot(x, n, as.cloud = FALSE, random.order = FALSE,
  rot.per = 0, ...)
```

Arguments

x	The important_terms object.
n	The number of rows to plot. If integer selects the frequency at the nth row and plots all rows >= that value. If proportional (less than 0) the frequency value for the nth% row is selected and plots all rows >= that value.
as.cloud	logical. If TRUE a wordcloud will be plotted rather than a bar plot.
random.order	logical. Should the words be place randomly around the cloud or if FALSE the more frequent words are in the center of the cloud.
rot.per	The precentage of rotated words.
...	Other arguments passed to wordcloud.

plot.ngram_collocations *Plots a ngram_collocations Object*

Description

Plots a ngram_collocations object.

Usage

```
## S3 method for class 'ngram_collocations'
plot(x, drop.redundant.yaxis.text = TRUE,
  plot = TRUE, ...)
```

Arguments

x	The ngram_collocations object.
drop.redundant.yaxis.text	logical. If TRUE the second y axis text/ticks, in the heat plot are dropped.
plot	logical. If TRUE the output is plotted.
...	ignored.

Value

Returns a list of the three **ggplot2** objects that make the combined plot.

`plot.tag_co_occurrence` *Plots a tag_co_occurrence Object*

Description

Plots a tag_co_occurrence object

Usage

```
## S3 method for class 'tag_co_occurrence'
plot(x, cor = FALSE, edge.weight = 5,
  node.weight = 5, edge.color = "gray80", node.color = "orange",
  bar.color = node.color, font.color = "gray55",
  bar.font.color = ifelse(bar, "gray96", bar.color),
  background.color = NULL, bar.font.size = TRUE, node.font.size = 3,
  digits = 1, min.edge.cutoff = 0.15, plot.widths = c(0.6, 0.4),
  bar = FALSE, type = "both", ...)
```

Arguments

`x`	A tag_co_occurrence object.
`cor`	logical. If `TRUE` the correlation matrix is used for the network graph, otherwise the adjacency matrix is used.
`edge.weight`	A weight for the edges.
`node.weight`	A weight for the nodes.
`edge.color`	A color for the edges.
`node.color`	A color for the nodes.
`bar.color`	A color for the bar fill; defaults to `node.color`.
`font.color`	A color for the node and axis text.
`bar.font.color`	A color for the bar/dotplot (mean co-occurrences).
`background.color`	The plot background color.
`bar.font.size`	A font size for the bar/dotplot (mean co-occurrences). Default tries to calculate based on number of bars.
`node.font.size`	The size for the node labels.
`digits`	The number of digits to print for bar/dotplot font (mean co-occurrences).
`min.edge.cutoff`	A minimum value to use as a cut-off in the network plot. If a value in the correlation/adjacency matrix is below this value, no edge will be plotted for the tag (node) connection.
`plot.widths`	A vector of proportions of length 2 and totalling 1 corresponding to the relative width of the network and bar/dotplot.
`bar`	logical. If `TRUE` a bar plot is used as the second plot, otherwise a bubble-dotplot is used.
`type`	The graph type (network & bar/dotplot). Choices are: `"bar"`, `"network"`, or `"both"` corresponding to the graph type to print.
`...`	ignored.

Value

Invisibly returns the network and dotplot/bar plot as a list.

`plot.term_count` *Plots a term_count object*

Description

Plots a term_count object.

Usage

```
## S3 method for class 'term_count'
plot(x, labels = FALSE, low = "white", high = "red",
  grid = NA, label.color = "grey70", label.size = 3, label.digits = if
  (weight == "count") {     0 } else {     2 }, weight = "percent", ...)
```

Arguments

`x`	The term_count object.
`labels`	logical. If `TRUE` the cell count values will be included on the heatmap.
`low`	The color to be used for lower values.
`high`	The color to be used for higher values.
`grid`	The color of the grid (Use `NA` to remove the grid).
`label.color`	The color to make labels if `labels = TRUE`.
`label.size`	The size to make labels if `labels = TRUE`.
`label.digits`	The number of digits to print if labels are printed.
`weight`	The weight to apply to the cell values for gradient fill. Currently the following are available: `"proportion"`, `"percent"`, and `"count"`. See `weight` for additional information.
`...`	ignored

`plot.term_loc` *Plots a term_loc Object*

Description

Plots a term_loc object.

Usage

```
## S3 method for class 'term_loc'
plot(x, as.cloud = FALSE, random.order = FALSE,
  rot.per = 0, ...)
```

Arguments

x	The term_loc object.
as.cloud	logical. If TRUE a wordcloud will be plotted rather than a bar plot.
random.order	logical. Should the words be place randomly around the cloud or if FALSE the more frequent words are in the center of the cloud.
rot.per	The precentage of rotated words.
...	Other arguments passed to wordcloud.

---

plot.token_count *Plots a token_count object*

Description

Plots a token_count object.

Usage

```
## S3 method for class 'token_count'
plot(x, labels = FALSE, low = "white", high = "red",
  grid = NA, label.color = "grey70", label.size = 3, label.digits = if
  (weight == "count") {     0 } else {     2 }, weight = "percent", ...)
```

Arguments

x	The token_count object.
labels	logical. If TRUE the cell count values will be included on the heatmap.
low	The color to be used for lower values.
high	The color to be used for higher values.
grid	The color of the grid (Use NA to remove the grid).
label.color	The color to make labels if labels = TRUE.
label.size	The size to make labels if labels = TRUE.
label.digits	The number of digits to print if labels are printed.
weight	The weight to apply to the cell values for gradient fill. Currently the following are available: "proportion", "percent", and "count". See weight for additional information.
...	ignored

plot.validate_model *Plots a validate_model Object*

Description

Plots a validate_model object

Usage

```
## S3 method for class 'validate_model'
plot(x, digits = 1, size = 0.65, height = 0.3,
  ...)
```

Arguments

x	A validate_model object.
digits	The number of digits to display n percents.
size	The size of error bars.
height	The height of error bars.
...	ignored.

plot_ca *Plot Term Count as Correspondence Analysis*

Description

A wrapper for the **ca**'s ca + plot or plot3d.ca functions for plotting a simple correspondence analysis.

Usage

```
plot_ca(x, D3 = TRUE, ...)
```

Arguments

x	A termco object.
D3	logical. If TRUE plots in 3-d.
...	Other arguments passed to plot and plot3d.ca.

Value

Plots a correspondence analysis.

Examples

```
data(presidential_debates_2012)

discoure_markers <- list(
    response_cries = c("\\boh", "\\bah", "\\baha", "\\bouch", "yuk"),
    back_channels = c("uh[- ]huh", "uhuh", "yeah"),
    summons = "hey",
    justification = "because"
)

(markers <- with(presidential_debates_2012,
    term_count(dialogue, list(person, time), discoure_markers)
))

plot_ca(markers, D3 = FALSE)
## Not run:
plot_ca(markers)

## End(Not run)
```

---

plot_counts *Horizontal Bar Plot of Group Counts*

Description

Plot the counts of groups within a vector or list as a horizontal bar plot.

Usage

```
plot_counts(x, n = NULL, percent = TRUE, item.name = "Terms",
  rev = FALSE, drop = TRUE, ...)
```

Arguments

x	A vector or list of elements.
n	Minimum frequency to be shown in the plot. If NULL all are shown.
percent	logical. If TRUE the x axis is scaled as percentages. Otherwise, the x axis is counts.
item.name	The name of the variable that contains the groups (different element in the vector/list).
rev	logical. If TRUE the bars go from least to greatest.
drop	logical. If FALSE and x is an as_terms object created from a term_count object, then unfound terms will not be dropped.
...	ignored.

Value

ggplot2 object.

Examples

```
x <- sample(LETTERS, 100, TRUE)
y <- lapply(1:100, function(i) sample(LETTERS[1:10], sample(0:5, 1), TRUE))
y <- sapply(y, function(x) {
    if(identical(x, character(0))) return(NULL)
    x
})

plot_counts(x)
plot_counts(y)

## Example
library(dplyr)
data(presidential_debates_2012)

discoure_markers <- list(
    response_cries = c("\\boh", "\\bah", "\\baha", "\\bouch", "yuk"),
    back_channels = c("uh[- ]huh", "uhuh", "yeah"),
    summons = "hey",
    justification = "because"
)

presidential_debates_2012 %>%
    with(., term_count(dialogue, TRUE, discoure_markers)) %>%
    as_terms() %>%
    plot_counts() +
        ggplot2::xlab("Tags")

presidential_debates_2012 %>%
    with(., term_count(dialogue, TRUE, discoure_markers)) %>%
    as_terms() %>%
    plot_freq(size=3) +
        ggplot2::xlab("Number of Tags")

presidential_debates_2012 %>%
    with(., term_count(dialogue, TRUE, discoure_markers)) %>%
    as_terms() %>%
    plot_counts(percent=FALSE, item.name = "Tags")
```

`plot_cum_percent` *Plot Cumulative Percent of Terms*

Description

Plot a cumulative percentage of terms for frequent terms.

Usage

```
plot_cum_percent(x, rotate.term = TRUE, ...)
```

Arguments

x	A `frequent_terms` object.

rotate.term	logical. If TRUE the term labels will be rotated 45 degrees.
...	ignored.

Examples

```
plot_cum_percent(frequent_terms(presidential_debates_2012[["dialogue"]]))
```

---

plot_freq *Vertical Bar Plot of Frequencies of Counts*

Description

Plot the counts of groups within a vector or list as a horizontal bar plot.

Usage

```
plot_freq(x, direct.label = TRUE, size = 4, label.diff, digits = 1,
  top.diff.weight = 0.06, ...)
```

Arguments

x	A vector or list of elements.
direct.label	logical. If TRUE count + percent labels are placed above bars.
size	The size to plot the text above the bars.
label.diff	The amount to place the labels above the bars. If missing a reasonable guess is attempted.
digits	The number of percent digits to print.
top.diff.weight	A weight to apply to the space between the top bar and the top of the plot area.
...	Other arguments passed to geom_text.

Value

ggplot2 object

Examples

```
x <- sample(LETTERS, 100, TRUE)
y <- lapply(1:100, function(i) sample(LETTERS[1:10], sample(0:5, 1), TRUE))
y <- sapply(y, function(x) {
    if(identical(x, character(0))) return(NULL)
    x
})

plot_freq(x)
plot_freq(y)

## Example
library(dplyr)
data(presidential_debates_2012)
```

```
discoure_markers <- list(
    response_cries = c("\\boh", "\\bah", "\\baha", "\\bouch", "yuk"),
    back_channels = c("uh[- ]huh", "uhuh", "yeah"),
    summons = "hey",
    justification = "because"
)

presidential_debates_2012 %>%
    with(., term_count(dialogue, TRUE, discoure_markers)) %>%
    as_terms() %>%
    plot_freq(size=3) +
        ggplot2::xlab("Number of Tags")

presidential_debates_2012 %>%
    with(., term_count(dialogue, TRUE, discoure_markers)) %>%
    as_terms() %>%
    plot_counts() +
        ggplot2::xlab("Tags")
```

`presidential_debates_2012`
2012 U.S. Presidential Debates

Description

A dataset containing a cleaned version of all three presidential debates for the 2012 election.

Usage

```
data(presidential_debates_2012)
```

Format

A data frame with 2912 rows and 4 variables

Details

- person. The speaker
- tot. Turn of talk
- dialogue. The words spoken
- time. Variable indicating which of the three debates the dialogue is from

print.as_terms *Prints an as_terms Object*

Description

Prints an as_terms object.

Usage

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

Arguments

x	The as_terms object.
...	ignored

print.combine_counts *Prints a combine_counts Object*

Description

Prints a combine_counts object.

Usage

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

Arguments

x	The combine_counts object.
...	ignored

print.coverage *Prints a coverage Object*

Description

Prints a coverage object

Usage

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

Arguments

x	The coverage object.
...	ignored

print.evaluate *Prints an evaluate Object*

Description

Prints an evaluate object

Usage

```
## S3 method for class 'evaluate'
print(x, digits = 3, ...)
```

Arguments

x	The evaluate object.
digits	The number of digits to print.
...	ignored

print.frequent_terms *Prints a frequent_terms Object*

Description

Prints a frequent_terms object.

Usage

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

Arguments

x	The frequent_terms object.
n	The number of rows to print. If integer selects the frequency at the nth row and prints all rows >= that value. If proportional (less than 0) the frequency value for the nth% row is selected and prints all rows >= that value.
...	ignored.

`print.frequent_terms_co_occurrence`
Prints a frequent_terms_co_occurrence Object

Description

Prints a frequent_terms_co_occurrence object

Usage

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

Arguments

`x`	A frequent_terms_co_occurrence object
`...`	Other arguments passed to `plot.tag_co_occurrence`.

`print.hierarchical_coverage`
Prints a hierarchical_coverage Object

Description

Prints a hierarchical_coverage object

Usage

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

Arguments

`x`	The hierarchical_coverage object.
`...`	ignored

print.hierarchical_coverage_regex *Prints a hierarchical_coverage_regex Object*

Description

Prints a hierarchical_coverage_regex object

Usage

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

Arguments

x	A hierarchical_coverage_regex object..
...	ignored.

print.important_terms *Prints an important_terms Object*

Description

Prints an important_terms object

Usage

```
## S3 method for class 'important_terms'
print(x, n = NULL, ...)
```

Arguments

x	An important_terms object.
n	The number of rows to print. If integer selects the frequency at the nth row and prints all rows >= that value. If proportional (less than 0) the frequency value for the nth% row is selected and prints all rows >= that value.
...	Ignored.

print.probe_list *Prints a probe_list Object*

Description

Prints a probe_list object

Usage

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

Arguments

x	A probe_list object
...	ignored.

print.search_term *Prints a search_term Object*

Description

Prints a search_term object.

Usage

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

Arguments

x	The search_term object.
...	ignored

print.split_data *Prints a split_data Object*

Description

Prints a split_data object

Usage

```
## S3 method for class 'split_data'
print(x, n = 6, ...)
```

Arguments

x	A split_data object
n	Number of elements to print
...	ignored.

print.summary.validate_model *Prints a summary.validate_model Object*

Description

Prints a summary.validate_model object

Usage

```
## S3 method for class 'summary.validate_model'
print(x, digits = 1, ...)
```

Arguments

x	A summary.validate_model object.
digits	The number of digits to display n percents.
...	ignored.

print.summary_hierarchical_coverage_regex *Prints a summary_hierarchical_coverage_regex Object*

Description

Prints a summary_hierarchical_coverage_regex object

Usage

```
## S3 method for class 'summary_hierarchical_coverage_regex'
print(x, digits = 1, ...)
```

Arguments

x	A summary_hierarchical_coverage_regex object.
digits	The number of digits to use in rounding percents.
...	ignored.

print.term_count *Prints a term_count Object*

Description

Prints a term_count object.

Usage

```
## S3 method for class 'term_count'
print(x, digits = 2, weight = "percent",
  zero.replace = "0", pretty = getOption("termco_pretty"), ...)
```

Arguments

x	The term_count object.
digits	The number of digits displayed.
weight	The weight type. Currently the following are available: "proportion", "percent". See weight for additional information.
zero.replace	The value to replace zero count elements with; defaults to "0".
pretty	logical. If TRUE the counts print in a pretty fashion, combining count and weighted information into a single display. pretty printing can be permanently removed with as_count.
...	ignored

print.token_count *Prints a token_count Object*

Description

Prints a token_count object.

Usage

```
## S3 method for class 'token_count'
print(x, digits = 2, weight = "percent",
  zero.replace = "0", pretty = getOption("termco_pretty"), ...)
```

Arguments

x	The token_count object.
digits	The number of digits displayed.
weight	The weight type. Currently the following are available: "proportion", "percent". See weight for additional information.
zero.replace	The value to replace zero count elements with; defaults to "0".
pretty	logical. If TRUE the counts print in a pretty fashion, combining count and weighted information into a single display. pretty printing can be permanently removed with as_count.
...	ignored

print.validate_model *Prints a validate_model Object*

Description

Prints a validate_model object

Usage

```
## S3 method for class 'validate_model'
print(x, digits = 1, ...)
```

Arguments

x	A validate_model object.
digits	The number of digits to display n percents.
...	ignored.

probe_colo_list *Generate List of Exploration* search_term_collocations *Function Calls*

Description

The task of determining the regexes used to feed a term_count object's term.list requires careful exploration of term use in context. This function generates a list of function calls for search_term_collocations (a wrapper for search_term + frequent_terms) with a user predefined data set and term list. This allows the user to explore a list of terms (such as from frequent_terms) and the accompanying terms that frequently collocate with these terms.

Usage

```
probe_colo_list(terms, data.name, copy2clip = getOption("termco.copy2clip"),
  ldots = "")
```

Arguments

terms	A vector of regex terms to explore (often populated from frequent_terms.
data.name	A character vector of a data set's name that will serve as the search context.
copy2clip	logical. If codeTRUE uses write_clip to copy the output to the clipboard. This option is most useful when trying to build a list regular expression model for easy pasting between testing a regex and putting it into the model. This argument can be set globally by setting options(termco.copy2clip = TRUE).
ldots	A string (starting with a comma) of additional arguments to include in the frequent_terms function of the list of function calls.

Value

Returns a string with the concatenated function calls. The print method separates the concatenated string into new line function calls. If `copy2clip = TRUE` the calls are easily pasted for use in exploration of the terms in the text data set.

See Also

Other probe functions: `probe_colo_plot_list`, `probe_list`

Examples

```
probe_colo_list(c("thank", "\\bthe", "ee"), "sam_i_am")
probe_colo_list(
    c("thank", "\\bthe", "ee"),
    "sam_i_am",
    ldots = ", n = 10, min.char = 5"
)

txt <- presidential_debates_2012[["dialogue"]]
terms <- frequent_terms(txt)[["term"]]
probe_colo_list(terms, "txt")

## Not run:
probe_colo_list(terms, "txt", copy2clip = TRUE)

## End(Not run)
```

`probe_colo_plot_list` *Generate List of Exploration* `search_term + frequent_terms + plot` *Function Calls*

Description

`probe_colo_plot_list` - The task of determining the regexes used to feed a `term_count` object's `term.list` requires careful exploration of term use in context. This function generates a list of function calls for `search_term + frequent_terms + plot` with a user predefined data set and term list. This allows the user to use bar plot explorations to explore a list of terms (such as from `frequent_terms`) and the accompanying terms that frequently collocate with these terms.

`probe_colo_plot` - Make the plots of `probe_colo_plot_list` directly to an external '`.pdf`' file.

Usage

```
probe_colo_plot_list(terms, data.name,
  copy2clip = getOption("termco.copy2clip"), ldots = "")

probe_colo_plot(terms, data, file = "Rplots.pdf", width = 5.5, height = 7,
  ...)
```

Arguments

terms	A vector of regex terms to explore (often populated from frequent_terms.
data.name	A character vector of a data set's name that will serve as the search context.
copy2clip	logical. If codeTRUE uses write_clip to copy the output to the clipboard. This option is most useful when trying to build a list regular expression model for easy pasting between testing a regex and putting it into the model. This argument can be set globally by setting options(termco.copy2clip = TRUE).
ldots	A string (starting with a comma) of additional arguments to include in the frequent_terms function of the list of function calls.
data	A vector of character strings.
file	A '.pdf' file to plot to.
width	The width of the graphics region in inches.
height	The height of the graphics region in inches.
...	Other arguments passed to frequent terms.

Value

Vprobe_colo_plot_list - Returns a string with the concatenated function calls. The print method separates the concatenated string into new line function calls. If copy2clip = TRUE the calls are easily pasted for use in exploration of the terms in the text data set.

Note

To actually make a '.pdf' file of the plots use the probe_colo_plot function directly. Also note that probe_colo_plot_list takes a character name for data.name whereas probe_colo_plot takes a an actual vector object for data.

See Also

Other probe functions: probe_colo_list, probe_list

Other probe functions: probe_colo_list, probe_list

Examples

```
probe_colo_plot_list(c("thank", "\\bthe", "ee"), "sam_i_am")
probe_colo_plot_list(
    c("thank", "\\bthe", "ee"),
    "sam_i_am",
    ldots = ", n = 10, min.char = 5"
)

txt <- presidential_debates_2012[["dialogue"]]
terms <- frequent_terms(txt)[["term"]]
probe_colo_plot_list(terms, "txt")

## Not run:
probe_colo_list(terms, "txt", copy2clip = TRUE)

#make an external file of plots
probe_colo_plot(terms, txt)

## End(Not run)
```

probe_list *Generate List of Exploration* search_term *Function Calls*

Description

The task of determining the regexes used to feed a term_count object's term.list requires careful exploration of term use in context. This function generates a list of function calls for search_term with a user predefined data set and term list.

Usage

```
probe_list(terms, data.name, copy2clip = getOption("termco.copy2clip"))
```

Arguments

terms	A vector of regex terms to explore (often populated from frequent_terms.
data.name	A character vector of a data set's name that will serve as the search context.
copy2clip	logical. If codeTRUE uses write_clip to copy the output to the clipboard. This option is most useful when trying to build a list regular expression model for easy pasting between testing a regex and putting it into the model. This argument can be set globally by setting options(termco.copy2clip = TRUE).

Value

Returns a string with the concatenated function calls. The print method separates the concatenated string into new line function calls. If copy2clip = TRUE the calls are easily pasted for use in exploration of the terms in the text data set.

See Also

Other probe functions: probe_colo_list, probe_colo_plot_list

Examples

```
probe_list(c("thank", "\\bthe", "ee"), "sam_i_am")

txt <- presidential_debates_2012[["dialogue"]]
terms <- frequent_terms(txt)[["term"]]
probe_list(terms, "txt")

## Not run:
probe_list(terms, "txt", copy2clip = TRUE)

## End(Not run)
```

sam_i_am *Sam I Am Text*

Description

A dataset containing a character vector of the text from Seuss's 'Sam I Am'.

Usage

```
data(sam_i_am)
```

Format

A character vector with 169 elements

References

Seuss, Dr. (1960). Green Eggs and Ham.

search_term *Search For Terms*

Description

`search_term` - Find text items that contain a term(s).

`search_term_which` - Find index of text items that contain a term(s).

Usage

```
search_term(text.var, term, exclude = NULL, and = NULL,
  ignore.case = TRUE, ...)

search_term_which(text.var, term, exclude = NULL, and = NULL,
  ignore.case = TRUE)
```

Arguments

`text.var`	A vector of character strings.
`term`	A regular expression to search for (uses `grep`).
`exclude`	A regular expression to exclude cases for (uses `grep`).
`and`	A regular expression that must also be contained in addition to `term` (uses `grep`).
`ignore.case`	logical. Should `grep` be done independent of case? Can also be length 3 corresponding to the arguments `term`, `exclude`, & `and`.
`...`	ignored.

Value

`search_term` - Returns a text vector meeting `term` regex but not `exclude` regex.

Examples

```
search_term_which(sam_i_am, "\\bsam")
search_term(sam_i_am, "\\bsam")
search_term(sam_i_am, c('green', "\\bsam"))
```

`search_term_collocations` *Search For Collocations*

Description

A wrapper for `search_term` + `frequent_terms`. Find words that frequently collocate with a term(s). Note that the 'term regexes are eliminated from the output of top occurring terms.

Usage

```
search_term_collocations(text.var, term, n = 10, ignore.case = TRUE, ...)
```

Arguments

`text.var`	A vector of character strings.
`term`	A regular expression(s) to search for (uses `grep`).
`n`	The number of rows to print. If integer selects the frequency at the nth row and prints all rows >= that value. If proportional (less than 0) the frequency value for the nth% row is selected and prints all rows >= that value.
`ignore.case`	logical. Should `grep` be done independent of case?
`...`	Other arguments passed to `search_term` and `frequent_terms`.

Value

Returns a `data.frame` of collocating terms and frequencies.

Author(s)

Steve T. Simpson and Tyler Rinker <tyler.rinker@gmail.com>.

Examples

```
## Example 1
search_term_collocations(sam_i_am, "\\bsam")
search_term_collocations(sam_i_am, c('green', "\\bsam"))
search_term_collocations(sam_i_am, c('green', "\\bsam"), min.char=2)

## Example 2
top_colo <- search_term_collocations(
    presidential_debates_2012[["dialogue"]],
    "president",
    n =50
)

top_colo
plot(top_colo)
```

```
plot(top_colo, as.cloud=TRUE)

## Example 3
top_colo_exclude <- search_term_collocations(
    presidential_debates_2012[["dialogue"]],
    "president",
    exclude = "obama",
    n =50
)

top_colo_exclude
plot(top_colo_exclude)
```

`split_data` *Split Data Into Training and Test*

Description

Split a data set into training and testing data.

Usage

```
split_data(data, n.train = 0.5, ...)

## S3 method for class 'data.frame'
split_data(data, n.train = 0.5, ...)

## Default S3 method:
split_data(data, n.train = 0.5, ...)
```

Arguments

`data`	A `data.frame` or `vector`.
`n.train`	An integer (number of) or proportion (proportion of) dictating how many observations to place in the training set.
`...`	ignored.

Value

Returns a named list of split data; a `train` data set and a `test` data set.

Examples

```
(split_dat <- split_data(mtcars))
split_dat$train
split_dat$test
split_data(mtcars, .8)

split_data(mtcars, 20)
split_data(LETTERS)
split_data(LETTERS, .4)
split_data(LETTERS, 10)
```

summary.hierarchical_coverage_regex *Summary of an hierarchical_coverage_regex Object*

Description

Summary of an hierarchical_coverage_regex object

Usage

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

Arguments

object	An hierarchical_coverage_regex object.
...	ignored.

summary.validate_model *Summary of an validate_model Object*

Description

Summary of an validate_model object

Usage

```
## S3 method for class 'validate_model'
summary(object, adjust.discrete = FALSE,
  ordered = TRUE, ...)
```

Arguments

object	An validate_model object.
adjust.discrete	logical. Should an additional ammount be deducted from the limits to account for dicrete data
ordered	logical. If TRUE the rows are ordered by tag accuracy.
...	ignored.

References

http://onlinestatbook.com/2/estimation/proportion_ci.html

tag_co_occurrence *Explore Tag Co-Occurrence*

Description

Explore tag co-occurrence. The resulting list comes with a plot method that allows the user to use a network graph to view the connections between tags as well as the average number of other tags that co-occur with each of the regex tags. This can provide information regarding the discriminatory power of each regex that corresponds to a tag.

Usage

```
tag_co_occurrence(x, ...)
```

Arguments

x	A term_count object.
...	ignored.

Value

Returns a list of:

ave_tag	A 2 column data.frame of tags and the average number of other tags that co-occur with it.
cor	A min-max scaled correlation matrix between tags; diagonals set to 0.
adjacency	An adjacency matrix between tags.
min_max_adjacency	A min-max scaled adjacency matrix between tags; diagonals set to 0.
node_size	The diagonals from the adjacency matrix; the number of times a tag occurred.

Author(s)

Steve T. Simpson and Tyler Rinker <tyler.rinker@gmail.com>.

Examples

```
## Not run:
## Example 1
regs <- frequent_terms(presidential_debates_2012[["dialogue"]])[[1]]
regs <- setNames(as.list(regs), regs)

model <- with(presidential_debates_2012,
    term_count(dialogue, TRUE, regs)
)

x <- tag_co_occurrence(model)
names(x)
setNames(
   lapply(names(x), function(a) {if(is.matrix(x[[a]])){ round(x[[a]], 2)} else { x[[a]] }}),
    names(x)
)
```

```
heatmap(x[["cor"]])
heatmap(x[["min_max_adjacency"]])
barplot(sort(x[["node_size"]], TRUE), las=2)
barplot(setNames(x[["ave_tag"]][[2]], x[["ave_tag"]][[1]]), las=2)

plot(x)
plot(x, cor=FALSE)
plot(x, min.edge.cutoff = .1, node.color = "#1CDB4F")
plot(x, min.edge.cutoff = .2, node.color = "gold", digits = 3)
plot(x, bar = TRUE)

##================================================
## Interactive chord diagram and network graph of
## tag co-occurrence
##================================================

## Load Required Add-on Packages
if (!require("pacman")) install.packages("pacman")
pacman::p_load(igraph, qrage)
pacman::p_load_gh("mattflor/chorddiag", "trinker/textshape")

## Matrix Manipulation Function
remove_diags <- function(mat, rm.lower = FALSE, order = TRUE, ...) {
    diag(mat) <- 0
    if (isTRUE(rm.lower)) mat[lower.tri(mat)] <- 0
    if (order) {
        ord <- order(rowSums(mat))
        mat <- mat[ord, ord]
    }
    mat
}

##--------------
## Chord Diagram
##--------------
chorddiag::chorddiag(
    remove_diags(x[["adjacency"]]),
    margin = 150,
    showTicks =FALSE,
    groupnamePadding = 5,
    groupThickness = .05,
    chordedgeColor = NA
)

##--------------
## Network Graph
##--------------
graph <- igraph::graph.adjacency(
    remove_diags(x[["adjacency"]], order=FALSE),
    weighted = TRUE
)

linkdf <- stats::setNames(get.data.frame(graph), c("source", "target", "value"))

qrage::qrage(
    links = linkdf,
    nodeValue = textshape::tidy_vector(x[['node_size']]),
```

```
    cut = 0.1
)

## Example 2
regs2 <- frequent_terms(presidential_debates_2012[["dialogue"]], n=50)[[1]]
regs2 <- setNames(as.list(regs2), regs2)

model2 <- with(presidential_debates_2012,
    term_count(dialogue, TRUE, regs2)
)

x2 <- tag_co_occurrence(model2)
plot(x2)
plot(x2, bar = FALSE, min.edge.cutoff = .13)
plot(x2, bar = FALSE, min.edge.cutoff = .18, node.color = "#ead453")
plot(x2, node.weight = 3)
plot(x2, edge.weight = 20, node.weight = 5)

plot(x2, edge.color = "gray80", node.color = "grey50", font.color = "white",
    background.color = "black")

## Small Number of Tags Example
plot(tag_co_occurrence(markers), node.weight = 5, min.edge.cutoff = .08)

## End(Not run)
```

`termco` *Counts of Terms and Substrings*

Description

A small suite of functions used to count terms and substrings in strings.

`term_before` *Extract Terms from Relative Locations*

Description

`term_before` - View the frequency of terms before a regex/term.

`term_after` - View the frequency of terms after a regex/term.

`term_first` - View the frequency of terms starting each string.

Usage

```
term_before(text.var, term, ignore.case = TRUE, ...)

term_after(text.var, term, ignore.case = TRUE, ...)

term_first(text.var, ignore.case = TRUE, ...)
```

Arguments

text.var	The text string variable.
term	A regex term to provide the search position.
ignore.case	logical. If FALSE, the pattern matching is case sensitive and if TRUE, case is ignored during matching.
...	ignored.

Value

Returns a data.frame of terms and frequencies

Examples

```
term_before(presidential_debates_2012$dialogue, 'president')
term_after(presidential_debates_2012$dialogue, 'president')
term_after(presidential_debates_2012$dialogue, 'oil')
term_first(presidential_debates_2012$dialogue)

x <- term_before(presidential_debates_2012$dialogue, 'president')
plot(x)

## Not run:
library(dplyr); library(lexicon)

pos_df_pronouns[['pronoun']][1:5] %>%
    lapply(function(x){
        term_after(presidential_debates_2012$dialogue, paste0("\\b", x, "\\b"))
    }) %>%
    setNames(pos_df_pronouns[['pronoun']][1:5])

term_first(presidential_debates_2012$dialogue) %>%
    filter(!term %in% tolower(sw_dolch) & !grepl("'", term))

## End(Not run)
```

---

term_cols *Get Term/Group Columns*

Description

Convenience functions to grab just the term or grouping variable columns from a term_count object.

Usage

```
term_cols(x, ...)

group_cols(x, ...)
```

Arguments

x	A term_count object.
...	ignored.

Value

Returns a tibble frame of just terms or grouping variables.

Examples

```
term_cols(markers)
group_cols(markers)
```

---

term_count *Search For and Count Terms*

Description

term_count - Search a string by any number of grouping variables for categories (themes) of grouped root terms/substrings.

Usage

```
term_count(text.var, grouping.var = NULL, term.list, ignore.case = TRUE,
  pretty = ifelse(isTRUE(grouping.var), FALSE, TRUE), group.names, ...)
```

Arguments

text.var	The text string variable.
grouping.var	The grouping variable(s). Default NULL generates one word list for all text. Also takes a single grouping variable or a list of 1 or more grouping variables. If TRUE an id variable is used with a seq_along the text.var.
term.list	A list of named character vectors. 'codeterm_count can be used in a hierarchical fashion as well; that is a list of regexes that can be passed and counted and then a second (or more) pass can be taken with a new set of regexes on only those rows/text elements that were left untagged (count rowSums is zero). This is accomplished by passing a list of lists of regexes. See **Examples** for the **hierarchical terms** section for a demonstration.
ignore.case	logical. If FALSE, the pattern matching is case sensitive and if TRUE, case is ignored during matching.
pretty	logical. If TRUE pretty printing is used. Pretty printing can be turned off globally by setting options(termco_pretty = FALSE).
group.names	A vector of names that corresponds to group. Generally for internal use.
...	ignored.

Value

Returns a tbl_df object of term counts by grouping variable.

Note

Note that while a term_count object prints as a combination of integer counts and weighted (default percent of terms) in parenthesis the underlying object is actually a tbl_df of integer term/substring counts. The user can alter a term_count object to print as integer permanently using the as_count function. A percent *Coverage* also prints. This is the rate of grouping variables with no term found (i.e., rowSums is zero for terms). For more details on coverage see coverage.

Examples

```
## Not run:
data(presidential_debates_2012)

discoure_markers <- list(
    response_cries = c("\\boh", "\\bah", "\\baha", "\\bouch", "yuk"),
    back_channels = c("uh[- ]huh", "uhuh", "yeah"),
    summons = "hey",
    justification = "because"
)

(markers <- with(presidential_debates_2012,
    term_count(dialogue, list(person, time), discoure_markers)
))

print(markers, pretty = FALSE)
print(markers, zero.replace = "_")
plot(markers)
plot(markers, labels=TRUE)

# permanently remove pretty printing
(markers2 <- as_count(markers))

# manipulating the output in a dplyr chain
library(dplyr)

presidential_debates_2012 %>%
    with(., term_count(dialogue, list(person, time), discoure_markers)) %>%
    as_count()  # removes pretty print method (not necessary to manipulate)

presidential_debates_2012 %>%
    with(., term_count(dialogue, list(person, time), discoure_markers)) %>%
    mutate(totals = response_cries + back_channels + summons + justification) %>%
    arrange(-totals)

## hierarchical terms
trms <- frequent_terms(presidential_debates_2012[["dialogue"]])[[1]]

discoure_markers <- list(
    response_cries = c("\\boh", "\\bah", "\\baha", "\\bouch", "yuk"),
    back_channels = c("uh[- ]huh", "uhuh", "yeah"),
    summons = "hey",
    justification = "because"
)

dbl_list <- list(
    discoure_markers,
    setNames(as.list(trms[1:8]), trms[1:8]),
    setNames(as.list(trms[9:length(trms)]), trms[9:length(trms)])
)

x <- with(presidential_debates_2012,
    term_count(dialogue, TRUE, dbl_list)
)

coverage(x)
```

```
## Auto mapping hierarchical terms w/ duplicate names
trpl_list <- list(
    list(
        response_cries = c("\\boh", "\\bah", "\\baha", "\\bouch", "yuk"),
        back_channels = c("uh[- ]huh", "uhuh", "yeah"),
        summons = "hey",
        justification = "because"
    ),
    list(summons ='the'),
    list(summons = 'it', justification = 'ed\\s')
)

(x2 <- with(presidential_debates_2012, term_count(dialogue, TRUE, trpl_list)))

## get the pre-collapse hierarchical coverage
attributes(x2)[['pre_collapse_coverage']]

## End(Not run)
```

---

token_count *Count Fixed Tokens*

Description

Count the occurrence of tokens within a vector of strings. This function differs from term_count in that term_count is regex based, allowing for fuzzy matching. This function only searches for lower cased tokens (words, number sequences, or punctuation). This counting function is faster but less flexible.

Usage

```
token_count(text.var, grouping.var = NULL, token.list, stem = FALSE,
  keep.punctuation = TRUE, pretty = ifelse(isTRUE(grouping.var), FALSE,
  TRUE), group.names, ...)
```

Arguments

text.var	The text string variable.
grouping.var	The grouping variable(s). Default NULL generates one word list for all text. Also takes a single grouping variable or a list of 1 or more grouping variables. If TRUE an id variable is used with a seq_along the text.var.
token.list	A list of named character vectors of tokens. Search will combine the counts for tokens supplied that are in the same vector. Tokens are defined as "^([a-z']+|[0-9.]+|[[:punct: and should conform to this standard. 'codetoken_count can be used in a hierarchical fashion as well; that is a list of tokens that can be passed and counted and then a second (or more) pass can be taken with a new set of tokens on only those rows/text elements that were left untagged (count rowSums is zero). This is accomplished by passing a list of lists of search tokens. See **Examples** for the **hierarchical tokens** section for a demonstration.
stem	logical. If TRUE the search is done after the terms have been stemmed.

keep.punctuation
: logical. If TRUE the punctuation marks are considered as tokens.

pretty
: logical. If TRUE pretty printing is used. Pretty printing can be turned off globally by setting options(termco_pretty = FALSE).

group.names
: A vector of names that corresponds to group. Generally for internal use.

...
: Other arguments passed to q_dtm.

Value

Returns a tbl_df object of term counts by grouping variable. Has all of the same features as a term_count object, meaning functions that work on a term_count object will operate on a a token_count object as well.

Examples

```
token_list <- list(
    person = c('sam', 'i'),
    place = c('here', 'house'),
    thing = c('boat', 'fox', 'rain', 'mouse', 'box', 'eggs', 'ham'),
    no_like = c('not like')
)

token_count(sam_i_am, grouping.var = TRUE, token.list = token_list)
token_count(sam_i_am, grouping.var = NULL, token.list = token_list)

x <- presidential_debates_2012[["dialogue"]]

bigrams <- apply(ngram_collocations(x)[, 1:2], 1, paste, collapse = " ")
bigram_model <- token_count(x, TRUE, token.list = as_term_list(bigrams))
as_dtm(bigram_model)

## Not run:
if (!require("pacman")) install.packages("pacman")
pacman::p_load(tidyverse, lexicon, textshape)

token_list <- lexicon::nrc_emotions %>%
    textshape::column_to_rownames() %>%
    t() %>%
    textshape::as_list()

presidential_debates_2012 %>%
     with(token_count(dialogue, TRUE, token_list))

presidential_debates_2012 %>%
     with(token_count(dialogue, list(person, time), token_list))

presidential_debates_2012 %>%
     with(token_count(dialogue, list(person, time), token_list)) %>%
     plot()

## End(Not run)

## hierarchical tokens
token_list <- list(
    list(
        person = c('sam', 'i')
```

```
    ),
    list(
        place = c('here', 'house'),
        thing = c('boat', 'fox', 'rain', 'mouse', 'box', 'eggs', 'ham')
    ),
    list(
        no_like = c('not like'),
        thing = c('train', 'goat')
    )
)

(x <- token_count(sam_i_am, grouping.var = TRUE, token.list = token_list))
attributes(x)[['pre_collapse_coverage']]
```

---

uncovered *Uncovered/Untagged Group Variable*

Description

uncovered - Get logical vector of uncovered data from the original text used to build the model.

get_uncovered - Get text vector from the uncovered data of the original text used to build the model.

Usage

```
uncovered(x, ...)

get_uncovered(x, ...)
```

Arguments

x	A term_count object.
...	ignored.

Value

uncovered - Returns logical indeices of untagged/uncovered group variables.

get_uncovered - Returns a vector of uncovered text from the original data set used to train the model.

Note

This is most useful when grouping.var = TRUE and an id variable was created that corresponds to the text variable. This allows the user to quickly grab the untagged text.

Examples

```
library(dplyr)

untagged <- presidential_debates_2012 %>%
     with(., term_count(dialogue, TRUE, list(manners = c("please|excuse|sorry")))) %>%
     uncovered() %>%
     {presidential_debates_2012[., "dialogue"]} %>%
     unlist(use.names = FALSE)

## Shorthand equivalent to code chunk above
untagged <- presidential_debates_2012 %>%
     with(., term_count(dialogue, TRUE, list(manners = c("please|excuse|sorry")))) %>%
     get_uncovered()

frequent_terms(untagged)
search_term(untagged, colo("romney", "governor"))

search_term(untagged, colo("people")) %>%
    frequent_terms()
```

---

unnest_term_list *Unnest a Nested Term List*

Description

Term lists can be stored as lists within a list for use in `termc_count` in a hierarchical fashion. This structure is not always useful and can be taken to a single nest via `unnest_term_list`. The function detects if the `term.list` is nested or not and then unnests only if needed, thus allowing it to be safely used on both nested and unnested `term.list`s.

Usage

```
unnest_term_list(term.list, ...)
```

Arguments

`term.list`	A list of named character vectors. 'codeterm_count can be used in a hierarchical fashion as well; that is a list of regexes can be passed and counted and then a second (or more) pass can be taken wit a new set of regexes on only those rows/text elements that were left untagged (count `rowSums` is zero). This is accomplished by passing a `list` of `list`s of regexes. See **Examples** for the **hierarchical terms** section for a demonstration.
`...`	ignored.

Value

Returns a `list` of one level.

Examples

```
x <- list(
    a = setNames(as.list(LETTERS[1:5]),LETTERS[1:5]),
    b = setNames(as.list(LETTERS[6:11]), LETTERS[6:11])
)
y <- list(a=LETTERS[1:11])
unnest_term_list(x)
unnest_term_list(y)
```

---

update_names *Rename a term_count Object's Term Columns*

Description

Safely rename a term_count object's term columns and attributes.

Usage

```
update_names(x, old, new)
```

Arguments

x	A term_count object.
old	A vector of the current names.
new	A vector of new names corresponding to the order of old names.

Value

Returns a renamed term_count object.

Examples

```
data(presidential_debates_2012)

discoure_markers <- list(
    response_cries = c("\\boh", "\\bah", "\\baha", "\\bouch", "yuk"),
    back_channels = c("uh[- ]huh", "uhuh", "yeah"),
    summons = "hey",
    justification = "because"
)

(markers <- with(presidential_debates_2012,
    term_count(dialogue, list(person, time), discoure_markers)
))


update_names(markers, old = c('back_channels', 'summons'), new = c('bcs', 's'))
update_names(markers, old = c('person'), new = c('people'))
update_names(markers, old = c('person', 'back_channels', 'summons'), new = c('people', 'bcs', 's'))
attributes(update_names(markers, old = c('back_channels', 'summons'), new = c('bcs', 's')))
```

validated *A Simulated* validate_model *Output*

Description

A dataset containing similated validate_model output.

Usage

```
data(validated)
```

Format

A data frame with 65 rows and 2 variables

Details

- tag. The tag.
- correct. Dummy coded if the tag correctly assessed the text.

validate_model *Manual Assessment of a Model*

Description

validate_model - Check how well a regex model is tagging using human interaction to assess the model.

assign_validation_task - Create human assignments to assess how well a model is functioning. The coder can use the correct column to assess how well the tag fits the text columns.

Usage

```
validate_model(x, n = 20, width = 50, tags = 1, ...)

assign_validation_task(x, n = 20, checks = 1, coders = "coder",
  out = NULL, as.list = TRUE, ...)
```

Arguments

x	A term_count model object (i.e., grouping.var = TRUE was used in term_count).
n	The number of samples to take from each regex tag assignment. Tags with less than n will use the full number available.
width	The width of the text display.
tags	The number of classifications per row/element to allow. Ties are broken probabilistically by default.
checks	The number of coders needed per tag assignment.
coders	A vector of coders to assign tasks to.

out	A directory name to create and output csv file(s) to.
as.list	logical. Should the assignments be dsplayed as a list of data.frame or as a single data.frame?
...	Other arguments passed to classify.

Value

validate_model - Returns a data.frame of the class 'validate_model'. Note that the pretty print is a tag summarized version of the model accuracy standard error, and confidence intervals.

assign_validation_task - Returns a data.frame/.csv or list of data.frames/.csvs. Columns in the data.frames include:

coder	The assgned coder (person for the task).
index	The row/element number of the text.
correct	A blank column for coders to dummy/logical code if the tag assignment for that text was accurate.
tag	The tag that was assigned to the text.
text	The text to which the tag was assigned.

Note

This function assigns tags using the classify function. One element may recieve multiple tags.

Examples

```
## Not run:
data(presidential_debates_2012)

discoure_markers <- list(
    response_cries = c("\\boh", "\\bah", "\\baha", "\\bouch", "yuk"),
    back_channels = c("uh[- ]huh", "uhuh", "yeah"),
    summons = "hey",
    justification = "because"
)

## A model (note: `grouping.var = TRUE` to make a model)
(x <- with(presidential_debates_2012,
    term_count(dialogue, grouping.var = TRUE, term.list = discoure_markers)
))

## Requires interaction
out <- validate_model(x)
out
plot(out)

## Assign tasks externally
assign_validation_task(x, checks = 3,
    coders = c('fred', 'jade', 'sally', 'jim', 'shelly'), out='testing')
assign_validation_task(x, checks = 3,
    coders = c('fred', 'jade', 'sally', 'jim', 'shelly'), as.list = FALSE,
    out='testing2')

## End(Not run)
```

weight *Weight Term Counts from* term_count

Description

Weight term counts from term_count object.

Usage

```
weight(x, weight = "percent", ...)

## S3 method for class 'term_count'
weight(x, weight = "percent", ...)

## S3 method for class 'token_count'
weight(x, weight = "percent", ...)
```

Arguments

x	A term_count object.
weight	A weight to use. Currently the following are available: "proportion", "percent".
...	ignored

Value

Returns a weighted tbl_df object of term counts by grouping variable.

Examples

```
library(dplyr)
data(markers)

weight(markers, "percent") %>%
    arrange(desc(n.words))

weight(markers, 'proportion')
```

Index

∗Topic **accuracy**
evaluate, 12
∗Topic **ca**
plot_ca, 29
∗Topic **classify**
classify, 6
∗Topic **collocate**
colo, 8
∗Topic **collocation**
search_term_collocations, 46
∗Topic **cooccur**
colo, 8
∗Topic **correspondence**
plot_ca, 29
∗Topic **count**
as_count, 3
∗Topic **coverage**
coverage, 10
hierarchical_coverage_regex, 18
hierarchical_coverage_term, 19
∗Topic **datasets**
markers, 21
presidential_debates_2012, 33
sam_i_am, 45
validated, 60
∗Topic **fit**
evaluate, 12
∗Topic **frequency**
frequent_terms, 13
ngram_collocations, 21
∗Topic **important**
important_terms, 20
∗Topic **model**
evaluate, 12
∗Topic **predict**
classify, 6
∗Topic **pretty**
as_count, 3
∗Topic **printing**
as_count, 3
∗Topic **project**
classification_project, 5
∗Topic **search**
search_term, 45
∗Topic **substring**
term_count, 53
∗Topic **term.list**
unnest_term_list, 58
∗Topic **term**
frequent_terms, 13
ngram_collocations, 21
term_count, 53
∗Topic **unnest**
unnest_term_list, 58
∗Topic **validate**
validate_model, 60
∗Topic **weight**
weight, 62
∗Topic **word**
frequent_terms, 13
ngram_collocations, 21

all_words *(*frequent_terms*)*, 13
as_count, 3, *4*, *40*, *53*
as_count<- *(*as_count*)*, 3
as_term_list, 4
as_terms, 4
assign_validation_task
*(*validate_model*)*, 60
attributes, *3*

classification_project, 5
classify, 6, *61*
collapse_tags, 7
collocations, *21, 22*
colo, 8
combine_counts, 9
coverage, 10, *53*

data.frame, *4*, *6*, *14*, *18–20*, *46, 47*

evaluate, 12

frequent_terms, 13, *15*, *46*
frequent_terms_co_occurrence, 15

geom_text, *32*
get_text, 17

get_uncovered *(*uncovered*)*, 57
grep, *45*
group_cols *(*term_cols*)*, 52

hierarchical_coverage_regex, 18, *19*
hierarchical_coverage_term, *18*, 19

important_terms, 20
is.global *(*classification_project*)*, 5

list, *6*, *12*, *53*, *55*, *58*

markers, 21
matrix, *4*, *6*
max.col, *6*

ngram_collocations, 21

package-termco *(*termco*)*, 51
plot.classify, 23
plot.frequent_terms, 23
plot.hierarchical_coverage_regex, 24
plot.hierarchical_coverage_term, 24
plot.important_terms, 25
plot.ngram_collocations, 25
plot.tag_co_occurrence, 26, *36*
plot.term_count, 27
plot.term_loc, 27
plot.token_count, 28
plot.validate_model, 29
plot_ca, 29
plot_counts, *23*, 30
plot_cum_percent, 31
plot_freq, 32
presidential_debates_2012, 33
print.as_terms, 34
print.combine_counts, 34
print.coverage, 34
print.evaluate, 35
print.frequent_terms, 35
print.frequent_terms_co_occurrence, 36
print.hierarchical_coverage, 36
print.hierarchical_coverage_regex, 37
print.important_terms, 37
print.probe_list, 38
print.search_term, 38
print.split_data, 38
print.summary.validate_model, 39
print.summary_hierarchical_coverage_regex, 39
print.term_count, 40
print.token_count, 40
print.validate_model, 41
probe_colo_list, 41, *43*, *44*
probe_colo_plot *(*probe_colo_plot_list*)*, 42
probe_colo_plot_list, *42*, 42, *44*
probe_list, *42*, *43*, 44

q_dtm, *56*

remove_stopwords, *20*
rowSums, *53*, *55*, *58*

sam_i_am, 45
sample, *6*
search_term, 45, *46*
search_term_collocations, 46
search_term_which *(*search_term*)*, 45
split_data, 47
summary.hierarchical_coverage_regex, 48
summary.validate_model, 48

tag_co_occurrence, *15*, 49
tbl_df, *53*, *56*, *62*
term_after *(*term_before*)*, 51
term_before, 51
term_cols, 52
term_count, *17*, *21*, *49*, *53*, 53, *55*, *57*, *62*
term_first *(*term_before*)*, 51
termco, 51
termco-package *(*termco*)*, 51
token_count, 55

uncovered, 57
unnest_term_list, 58
update_names, 59

validate_model, 60
validated, 60
vector, *6*, *12*, *47*

weight, *27*, *28*, *40*, 62
wordcloud, *23*, *25*, *28*
wordStem, *14*, *20*, *22*
write_clip, *8*, *41*, *43*, *44*