

Package ‘sentimentr’

October 20, 2018

Title Calculate Text Polarity Sentiment

Version 2.6.1

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

Description Calculate text polarity sentiment at the sentence level and optionally aggregate by rows or grouping variable(s).

Depends R (>= 3.4.0)

Suggests testthat

Imports data.table, ggplot2, graphics, grid, lexicon (>= 1.0.0), methods, stats, stringi, syuzhet, textclean (>= 0.6.1), textshape (>= 1.3.0), utils

License MIT + file LICENSE

LazyData TRUE

RoxygenNote 6.1.0.9000

URL <http://github.com/trinker/sentimentr>

BugReports <http://github.com/trinker/sentimentr/issues>

NeedsCompilation no

Author Tyler Rinker [aut, cre]

Repository CRAN

Date/Publication 2018-10-20 14:20:03 UTC

R topics documented:

as_key	2
available_data	5
average_downweighted_zero	7
combine_data	8
course_evaluations	9
crowdfunder_deflategate	9
crowdfunder_products	10
crowdfunder_self_driving_cars	11

crowdflower_weather	11
extract_profanity_terms	12
extract_sentiment_terms	13
general_rescale	14
get_sentences	15
get_sentences2	16
highlight	16
hotel_reviews	17
hu_liu_apex_reviews	18
hu_liu_cannon_reviews	19
hu_liu_jukebox_reviews	19
hu_liu_nikon_reviews	20
hu_liu_nokia_reviews	21
kaggle_movie_reviews	22
kotzias_reviews_amazon_cells	22
kotzias_reviews_imdb	23
kotzias_reviews_yelp	24
nyt_articles	25
plot.profanity	26
plot.profanity_by	26
plot.sentiment	27
plot.sentiment_by	28
presidential_debates_2012	28
print.extract_profanity_terms	29
print.extract_sentiment_terms	29
print.validate_sentiment	30
profanity	30
profanity_by	31
sam_i_am	33
sentiment	34
sentimentr	40
sentiment_attributes	40
sentiment_by	41
uncombine	44
validate_sentiment	45

Index **48**

as_key	<i>Create/Manipulate Hash Keys</i>
--------	------------------------------------

Description

as_key - Create your own hash keys from a data frame for use in key arguments such as polarity_dt in the sentiment function.

update_key - Add/remove terms to a current key.

update_polarity_table - Wrapper for update_key specifically for updating polarity tables.

update_valence_shifter_table - Wrapper for update_key specifically for updating valence shifter tables.

is_key - Logical check if an object is a key.

Usage

```
as_key(x, comparison = lexicon::hash_valence_shifters,
      sentiment = TRUE, ...)
```

```
update_key(key, drop = NULL, x = NULL,
          comparison = lexicon::hash_valence_shifters, sentiment = FALSE, ...)
```

```
update_polarity_table(key, drop = NULL, x = NULL,
                    comparison = lexicon::hash_valence_shifters, sentiment = FALSE, ...)
```

```
update_valence_shifter_table(key, drop = NULL, x = NULL,
                             comparison = lexicon::hash_sentiment_jockers_rinker,
                             sentiment = FALSE, ...)
```

```
is_key(key, sentiment = TRUE)
```

Arguments

x	A data.frame with the first column containing polarized words and the second containing polarity values.
comparison	A data.frame to compare to x. If elements in x's column 1 matches comparison's column 1 the accompanying row will be removed from x. This is useful to ensure polarity_dt words are not also found in valence_shifters_dt in sentiment . Use comparison = NULL to skip this comparison.
sentiment	logical. If TRUE checking expects column 2 of the input keys/ data.frame are expected to be numeric.
key	A sentimentr hash key.
drop	A vector of terms to drop.
...	ignored.

Details

For updating keys via update_key note that a polarity_dt and valence_shifters_dt are the primary dictionary keys used in the **sentimentr** package. The polarity_dt takes a 2 column data.frame (named x and y) with the first column being character and containing the words and the second column being numeric values that are positive or negative. valence_shifters_dt takes a 2 column data.frame (named x and y) with the first column being character and containing the words and the second column being integer corresponding to: (1) negators, (2) amplifiers, (3) de-amplifiers, and (4) diversative conjunctions (i.e., 'but', 'however', and 'although'). Also, note that if you are updating a valence_shifters_dt you need an appropriate comparison; most likely, comparison = sentimentr::polarity_dt.

Value

Returns a **data.table** object that can be used as a hash key.

Examples

```
key <- data.frame(
  words = sample(letters),
  polarity = rnorm(26),
  stringsAsFactors = FALSE
)

(mykey <- as_key(key))

## Looking up values
mykey[c("a", "k")][[2]]

## Drop terms from key
update_key(mykey, drop = c("f", "h"))

## Add terms to key
update_key(mykey, x = data.frame(x = c("dog", "cat"), y = c(1, -1)))

## Add terms & drop to/from a key
update_key(mykey, drop = c("f", "h"), x = data.frame(x = c("dog", "cat"), y = c(1, -1)))

## Explicitly key type (wrapper for `update_key` for sentiment table.
## See `update_valence_shifter_table` a corresponding valence shifter updater.
library(lexicon)
updated_hash_sentiment <- sentimentr::update_polarity_table(lexicon::hash_sentiment_huliu,
  x = data.frame(
    words = c('frickin', 'hairy'),
    polarity = c(-1, -1),
    stringsAsFactors = FALSE
  )
)

## Checking if you have a key
is_key(mykey)
is_key(key)
is_key(mtcars)
is_key(update_key(mykey, drop = c("f", "h")))

## Using syuzhet's sentiment lexicons
## Not run:
library(syuzhet)
(bing_key <- as_key(syuzhet::bing))
as_key(syuzhet::afinn)
as_key(syuzhet::syuzhet_dict)

sam <- gsub("Sam-I-am", "Sam I am", sam_i_am)
sentiment(sam, , polarity_dt = bing_key)
```

```

## The nrc dictionary in syuzhet requires a bit of data wrangling before it
## is in the correct shape to convert to a key.

library(syuzhet)
library(tidyverse)

nrc_key <- syuzhet:::nrc %>%
  dplyr::filter(
    sentiment %in% c('positive', 'negative'),
    lang == 'english'
  ) %>%
  dplyr::select(-lang) %>%
  mutate(value = ifelse(sentiment == 'negative', value * -1, value)) %>%
  dplyr::group_by(word) %>%
  dplyr::summarize(y = mean(value)) %>%
  sentimentr::as_key()

sentiment(sam, polarity_dt = nrc_key)

## The lexicon package contains a preformatted nrc sentiment hash table that
## can be used instead.
sentiment(sam, polarity_dt = lexicon::hash_sentiment_nrc)

## End(Not run)

## Using 2 vectors of words
## Not run:
install.packages("tm.lexicon.GeneralInquirer", repos="http://datacube.wu.ac.at", type="source")
require("tm.lexicon.GeneralInquirer")

positive <- terms_in_General_Inquirer_categories("Positiv")
negative <- terms_in_General_Inquirer_categories("Negativ")

geninq <- data.frame(
  x = c(positive, negative),
  y = c(rep(1, length(positive)), rep(-1, length(negative))),
  stringsAsFactors = FALSE
) %>%
  as_key()

geninq_pol <- with(presidential_debates_2012,
  sentiment_by(dialogue,
    person,
    polarity_dt = geninq
  ))

geninq_pol %>% plot()

## End(Not run)

```

Description

See available **sentimentr** data a data.frame. Note that `sentimentr_data` is the main function to be used but `available_data` is exposed to allow other packages to use the functionality in a generic way.

Usage

```
available_data(regex = NULL, package = "sentimentr", ...)

sentimentr_data(regex = NULL, package = "sentimentr", ...)
```

Arguments

<code>regex</code>	A regex to search for within the data columns.
<code>package</code>	The name of the package to extract data from.
<code>...</code>	Other arguments passed to <code>grep</code> .

Value

Returns a data.frame

Examples

```
sentimentr_data()
available_data() ## generic version for export
available_data(package = 'datasets')
sentimentr_data('^hu')
sentimentr_data('(hu|kot)')
combine_data(sentimentr_data('(hu|kot)')[[1]])

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

sentiment_data <- sentimentr_data('^hu') %>%
  pull(Data) %>%
  combine_data() %>%
  mutate(id = seq_len(n())) %>%
  as_tibble()

sentiment_test <- sentiment_data %>%
  select(-sentiment) %>%
  get_sentences() %$%
  sentiment(., by = c('id'))

testing <- sentiment_data %>%
  left_join(sentiment_test, by = 'id') %>%
  as_tibble() %>%
  mutate(
    actual = sign(sentiment),
    predicted = sign(ave_sentiment)
```

```

)

testing %%
  ftable(predicted, actual)

## End(Not run)

```

average_downweighted_zero

Downweighted Zeros Averaging

Description

average_downweighted_zero- Downweight the zeros in a vector for averaging. This is useful in the context of language where we don't want the neutral sentences to have such a strong influence on the general sentiment of the discourse with multiple sentences. Essentially, this means neutral sentences are seen as having less emotional impact than a polarized sentence.

average_weighted_mixed_sentiment- Upweight the negative values in a vector while also down-weighting the zeros in a vector. Useful for small text chunks with several sentences in which some one states a negative sentence but then uses the social convention of several positive sentences in an attempt to negate the impact of the negative. The affective state isn't a neutral but a slightly lessened negative state.

average_mean- Standard mean averaging with na.rm set to TRUE.

Usage

```
average_downweighted_zero(x, na.rm = TRUE, ...)
```

```
average_weighted_mixed_sentiment(x, mixed.less.than.zero.weight = 4,
  na.rm = TRUE, ...)
```

```
average_mean(x, na.rm = TRUE, ...)
```

Arguments

x	A numeric vector.
na.rm	logical. Should NA values should be stripped before the computation proceeds.
mixed.less.than.zero.weight	The weighting factor to multiply the negative elements of the vector by (this increases the intensity of the negatives in the numerator of the mean formula).
...	ignored.

Value

Returns a scalar summary of the re-weighted average

Examples

```
x <- c(1, 2, 0, 0, 0, -1)
mean(x)
average_downweighted_zero(x)
average_downweighted_zero(c(NA, x))
mean(c(0, 0, 0, x))
average_downweighted_zero(c(0, 0, 0, x))
```

combine_data

*Combine **sentimentr**'s Sentiment Data Sets*

Description

Combine trusted sentiment data sets from **sentimentr**.

Usage

```
combine_data(data = c("course_evaluations", "hotel_reviews",
  "kaggle_movie_reviews", "kotzias_reviews_amazon_cells",
  "kotzias_reviews_imdb", "kotzias_reviews_yelp", "nyt_articles"), ...)
```

Arguments

data	A character vector of sentimentr data sets.
...	ignored.

Value

Returns an rbinded **data.table** of sentiment data with the source added as column.

Examples

```
combine_data()
combine_data(c("kotzias_reviews_amazon_cells", "kotzias_reviews_imdb",
  "kotzias_reviews_yelp"))
```

course_evaluations *Student Course Evaluation Comments*

Description

A dataset containing a subset of comments and rating from Welch & Mihalcea's (2017) data set filtered to include comments with a one or more unambiguous sentiment rating.

Usage

```
data(course_evaluations)
```

Format

A data frame with 566 rows and 2 variables

Details

- sentiment. A numeric sentiment score
- text. The text from the evaluation

References

Welch, C. and Mihalcea, R. (2017). Targeted sentiment to understand student comments. In Proceedings of the International Conference on Computational Linguistics (COLING 2016).

<http://web.eecs.umich.edu/~mihalcea/downloads.html#GroundedEmotions>

crowdflower_deflategate

Twitter Tweets About the Deflategate

Description

A dataset containing Twitter tweets about Tom Brady's deflated ball scandal, taken from Crowdflower.

Usage

```
data(crowdflower_deflategate)
```

Format

A data frame with 11,786 rows and 2 variables

Details

- sentiment. A human scoring of the text.
- text. The sentences from the tweet.

References

<https://www.crowdflower.com/data-for-everyone>

crowdflower_products *Twitter Tweets About the Products*

Description

A dataset containing Twitter tweets about various products, taken from Crowdflower.

Usage

```
data(crowdflower_products)
```

Format

A data frame with 3,548 rows and 2 variables

Details

- sentiment. A human scoring of the text.
- text. The sentences from the tweet.

References

Cavender-Bares, K., (2013). Judge emotion about brands & products.

<https://www.crowdflower.com/data-for-everyone>

crowdflower_self_driving_cars

Twitter Tweets About Self Driving Cars

Description

A dataset containing Twitter tweets about self driving cars, taken from Crowdflower.

Usage

```
data(crowdflower_self_driving_cars)
```

Format

A data frame with 6,943 rows and 2 variables

Details

- sentiment. A human scoring of the text.
- text. The sentences from the tweet.

References

<https://www.crowdflower.com/data-for-everyone>

crowdflower_weather

Twitter Tweets About the Weather

Description

A dataset containing Twitter tweets about the weather, taken from Crowdflower.

Usage

```
data(crowdflower_weather)
```

Format

A data frame with 763 rows and 2 variables

Details

- sentiment. A human scoring of the text.
- text. The sentences from the tweet.

References

<https://www.crowdflower.com/data-for-everyone>

`extract_profanity_terms`*Extract Profanity Words*

Description

Extract the profanity words from a text.

Usage

```
extract_profanity_terms(text.var,  
  profanity_list = lexicon::profanity_alvarez, ...)
```

Arguments

<code>text.var</code>	The text variable. Can be a <code>get_sentences</code> object or a raw character vector though <code>get_sentences</code> is preferred as it avoids the repeated cost of doing sentence boundary disambiguation every time profanity is run.
<code>profanity_list</code>	A atomic character vector of profane words. The lexicon package has lists that can be used, including: <ul style="list-style-type: none">• <code>lexicon::profanity_alvarez</code>• <code>lexicon::profanity_arr_bad</code>• <code>lexicon::profanity_banned</code>• <code>lexicon::profanity_zac_anger</code>
<code>...</code>	Ignored.

Value

Returns a **data.table** with a columns of profane terms.

Examples

```
## Not run:  
bw <- sample(lexicon::profanity_alvarez, 4)  
mytext <- c(  
  sprintf('do you like this %s? It is %s. But I hate really bad dogs', bw[1], bw[2]),  
  'I am the best friend.',  
  NA,  
  sprintf('I %s hate this %s', bw[3], bw[4]),  
  "Do you really like it? I'm not happy"  
)  
  
x <- get_sentences(mytext)  
profanity(x)  
  
prof_words <- extract_profanity_terms(x)
```

```

prof_words
prof_words$sentence
prof_words$neutral
prof_words$profanity
data.table::as.data.table(prof_words)

attributes(extract_profanity_terms(x))$counts
attributes(extract_profanity_terms(x))$elements

brady <- get_sentences(crowdflower_deflategate)
brady_swears <- extract_profanity_terms(brady)

attributes(extract_profanity_terms(brady))$counts
attributes(extract_profanity_terms(brady))$elements

## End(Not run)

```

```
extract_sentiment_terms
```

Extract Sentiment Words

Description

Extract the sentiment words from a text.

Usage

```
extract_sentiment_terms(text.var,
  polarity_dt = lexicon::hash_sentiment_jockers_rinker, hyphen = "",
  ...)
```

Arguments

text.var	The text variable.
polarity_dt	A data.table of positive/negative words and weights with x and y as column names.
hyphen	The character string to replace hyphens with. Default replaces with nothing so 'sugar-free' becomes 'sugarfree'. Setting hyphen = " " would result in a space between words (e.g., 'sugar free').
...	Ignored.

Value

Returns a **data.table** with columns of positive and negative terms.

Examples

```

library(data.table)
set.seed(10)
x <- get_sentences(sample(hu_liu_cannon_reviews[[2]], 1000, TRUE))
sentiment(x)

pol_words <- extract_sentiment_terms(x)
pol_words
pol_words$sentence
pol_words$neutral
data.table::as.data.table(pol_words)

attributes(extract_sentiment_terms(x))$counts
attributes(extract_sentiment_terms(x))$elements

```

general_rescale	<i>Rescale a Numeric Vector</i>
-----------------	---------------------------------

Description

Rescale a numeric vector with the option to make signed (-1, 1, or 0) and retain zero as neutral.

Usage

```

general_rescale(x, lower = -1, upper = 1, mute = NULL,
  keep.zero = lower < 0, sign = FALSE, ...)

```

Arguments

x	A numeric vector.
lower	An upper limit to rescale to.
upper	A lower limit to rescale to.
mute	A positive value greater than 1 to lower the extremes and pull the fractions up. This becomes the denominator in a power to raise each element by (sign is retained) where the numerator is 1. This is useful for mellowing out the extremes.
keep.zero	logical. If TRUE the zeros are kept as neutral.
sign	logical. If TRUE the vector will be scaled as (-1, 1, or 0)
...	ignored.

Value

Returns a rescaled vector of the same length as x.

Examples

```

general_rescale(c(1, 0, -1))
general_rescale(c(1, 0, -1, 1.4, -2))
general_rescale(c(1, 0, -1, 1.4, -2), lower = 0, upper = 1)
general_rescale(c(NA, -4:3))
general_rescale(c(NA, -4:3), keep.zero = FALSE)
general_rescale(c(NA, -4:3), keep.zero = FALSE, lower = 0, upper = 100)

## mute extreme values
set.seed(10)
x <- sort(c(NA, -100, -10, 0, rnorm(10, 0, .1), 10, 100), na.last = FALSE)
general_rescale(x)
general_rescale(x, mute = 5)
general_rescale(x, mute = 10)
general_rescale(x, mute = 100)

```

get_sentences

Get Sentences

Description

get_sentences - Get sentences from a character vector, sentiment, or sentiment_by object.

Usage

```
get_sentences(x, ...)
```

Arguments

x A character vector, sentiment, or sentiment_by object.
... Other arguments passed to [split_sentence](#).

Value

Returns a list of vectors of sentences.

Examples

```

dat <- data.frame(
  w = c('Person 1', 'Person 2'),
  x = c(paste0(
    "Mr. Brown comes! He says hello. i give him coffee. i will ",
    "go at 5 p. m. eastern time. Or somewhere in between!go there"
  ), "One more thought for the road! I am going now. Good day."),
  y = state.name[c(32, 38)],
  z = c(.456, .124),
  stringsAsFactors = FALSE
)

```

```
get_sentences(dat$x)
get_sentences(dat)
```

get_sentences2	<i>Get Sentences (Deprecated)</i>
----------------	-----------------------------------

Description

Deprecated, use [get_sentences](#) instead.

Usage

```
get_sentences2(x, ...)
```

Arguments

x	A character vector, sentiment, or sentiment_by object.
...	Ignored.

highlight	<i>Polarity Text Highlighting</i>
-----------	-----------------------------------

Description

Highlight sentences within elements (row IDs) by sentiment polarity (positive = green; negative = pink) as an html file.

Usage

```
highlight(x, file = file.path(tempdir(), "polarity.html"), open = TRUE,
  digits = 3, ...)
```

Arguments

x	A sentiment_by object.
file	A name of the html file output.
open	logical. If TRUE the text highlighting document will attempt to be opened.
digits	The number of digits to print for each row level average sentiment score.
...	Ignored.

Value

Generates an html document with text highlighting.

Examples

```

library(data.table)
dat <- presidential_debates_2012
setDT(dat)

dat[, gr:={gr= paste(person, time); cumsum(c(TRUE, gr[-1]!= gr[-.N]))}]
dat <- dat[, list(person=person[1L], time=time[1L], dialogue=paste(dialogue,
  collapse = ' ')), by = gr][,gr:= NULL][,
  dialogue_split := get_sentences(dialogue)][]

(sent_dat <- with(dat, sentiment_by(dialogue_split, list(person, time))))

## Not run:
highlight(sent_dat)

## tidy approach
library(dplyr)
library(magrittr)

hu_liu_cannon_reviews %>%
  filter(review_id %in% sample(unique(review_id), 3)) %>%
  mutate(review = get_sentences(text)) %>%
  sentiment_by(review, review_id) %>%
  highlight()

## End(Not run)

```

 hotel_reviews

Hotel Reviews

Description

A dataset containing a random sample (n = 5000 of 1,621,956) of Wang, Lu, & Zhai's (2011) hotel reviews data set scraped by the authors from <http://www.tripadvisor.com>.

Usage

```
data(hotel_reviews)
```

Format

A data frame with 5000 rows and 2 variables

Details

- sentiment. The overall rating for the experience
- text. The text review of the hotel

References

Wang, H., Lu, Y., and Zhai, C. (2011). Latent aspect rating analysis without aspect keyword supervision. In Proceedings of the 17th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'2011), 618-626.

Original URL: '<http://sifaka.cs.uiuc.edu/~wang296/Data/index.html>'

hu_liu_apex_reviews *Apex AD2600 Progressive-scan DVD player Product Reviews From Amazon*

Description

A dataset containing Amazon product reviews for the Apex AD2600 Progressive-scan DVD player. This data set was compiled by Hu and Liu (2004). Where a sentence contains more than one opinion score and average of all scores is used.

Usage

```
data(hu_liu_apex_reviews)
```

Format

A data frame with 740 rows and 3 variables

Details

- sentiment. Hu and Liu (2004)'s average opinion rating for a sentence. Negative and positive reflects direction, a negative or positive sentiment. Opinion strength varies between 3 (strongest), and 1 (weakest). number. The review number.
- text. The text from the review.
- review_id. The review number.

References

Minqing Hu and Bing Liu. (2004). Mining and summarizing customer reviews. Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD-04).

Minqing Hu and Bing Liu. (2004). "Mining Opinion Features in Customer Reviews. Proceedings of Nineteenth National Conference on Artificial Intelligence (AAAI-2004).

'<https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html>'

hu_liu_cannon_reviews *Cannon G3 Camera Product Reviews From Amazon*

Description

A dataset containing Amazon product reviews for the Cannon G3 Camera. This data set was compiled by Hu and Liu (2004). Where a sentence contains more than one opinion score and average of all scores is used.

Usage

```
data(hu_liu_cannon_reviews)
```

Format

A data frame with 597 rows and 3 variables

Details

- sentiment. Hu and Liu (2004)'s average opinion rating for a sentence. Negative and positive reflects direction, a negative or positive sentiment. Opinion strength varies between 3 (strongest), and 1 (weakest). number. The review number.
- text. The text from the review.
- review_id. The review number.

References

Minqing Hu and Bing Liu. (2004). Mining and summarizing customer reviews. Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD-04).

Minqing Hu and Bing Liu. (2004). "Mining Opinion Features in Customer Reviews. Proceedings of Nineteenth National Conference on Artificial Intelligence (AAAI-2004).

'<https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html>'

hu_liu_jukebox_reviews

Creative Labs Nomad Jukebox Zen Xtra 40GB Product Reviews From Amazon

Description

A dataset containing Amazon product reviews for the Creative Labs Nomad Jukebox Zen Xtra 40GB. This data set was compiled by Hu and Liu (2004). Where a sentence contains more than one opinion score and average of all scores is used.

Usage

```
data(hu_liu_jukebox_reviews)
```

Format

A data frame with 1716 rows and 3 variables

Details

- `sentiment`. Hu and Liu (2004)'s average opinion rating for a sentence. Negative and positive reflects direction, a negative or positive sentiment. Opinion strength varies between 3 (strongest), and 1 (weakest). `number`. The review number.
- `text`. The text from the review.
- `review_id`. The review number.

References

Minqing Hu and Bing Liu. (2004). Mining and summarizing customer reviews. Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD-04).

Minqing Hu and Bing Liu. (2004). "Mining Opinion Features in Customer Reviews. Proceedings of Nineteenth National Conference on Artificial Intelligence (AAAI-2004).

'<https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html>'

hu_liu_nikon_reviews *Nikon Coolpix 4300 Product Reviews From Amazon*

Description

A dataset containing Amazon product reviews for the Nikon Coolpix 4300. This data set was compiled by Hu and Liu (2004). Where a sentence contains more than one opinion score and average of all scores is used.

Usage

```
data(hu_liu_nikon_reviews)
```

Format

A data frame with 346 rows and 3 variables

Details

- `sentiment`. Hu and Liu (2004)'s average opinion rating for a sentence. Negative and positive reflects direction, a negative or positive sentiment. Opinion strength varies between 3 (strongest), and 1 (weakest). `number`. The review number.
- `text`. The text from the review.
- `review_id`. The review number.

References

Minqing Hu and Bing Liu. (2004). Mining and summarizing customer reviews. Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD-04).

Minqing Hu and Bing Liu. (2004)."Mining Opinion Features in Customer Reviews. Proceedings of Nineteenth National Conference on Artificial Intelligence (AAAI-2004).

'<https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html>'

hu_liu_nokia_reviews *Nokia 6610 Product Reviews From Amazon*

Description

A dataset containing Amazon product reviews for the Nokia 6610. This data set was compiled by Hu and Liu (2004). Where a sentence contains more than one opinion score and average of all scores is used.

Usage

```
data(hu_liu_nokia_reviews)
```

Format

A data frame with 546 rows and 3 variables

Details

- sentiment. Hu and Liu (2004)'s average opinion rating for a sentence. Negative and positive reflects direction, a negative or positive sentiment. Opinion strength varies between 3 (strongest), and 1 (weakest).
- number. The review number.
- text. The text from the review.
- review_id. The review number.

References

Minqing Hu and Bing Liu. (2004). Mining and summarizing customer reviews. Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD-04).

Minqing Hu and Bing Liu. (2004)."Mining Opinion Features in Customer Reviews. Proceedings of Nineteenth National Conference on Artificial Intelligence (AAAI-2004).

'<https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html>'

kaggle_movie_reviews *Movie Reviews*

Description

A dataset containing sentiment scored movie reviews from a Kaggle competition posted by University of Michigan SI650. The data was originally collected from opinmind.com.

Usage

```
data(kaggle_movie_reviews)
```

Format

A data frame with 7,086 rows and 2 variables

Details

- sentiment. A numeric sentiment score
- text. The text from the review

References

<https://www.kaggle.com/c/si650winter11/data>

kotzias_reviews_amazon_cells

Kotzias Reviews: Amazon Cells

Description

A dataset containing a list of 4 review data sets. Each data set contains sentences with a positive (1) or negative review (-1) taken from reviews of products, movies, & restaurants. The data, compiled by Kotzias, Denil, De Freitas, & Smyth (2015), was originally taken from amazon.com, imdb.com, & yelp.com. Kotzias et al. (2015) provide the following description in the README: "For each website, there exist 500 positive and 500 negative sentences. Those were selected randomly for larger datasets of reviews. We attempted to select sentences that have a clearly positive or negative connotation [sic], the goal was for no neutral sentences to be selected. This data set has been manipulated from the original to be split apart by element (sentence split). The original 0/1 metric has also been converted to -1/1. Please cite Kotzias et al. (2015) if you reuse the data here.

Usage

```
data(kotzias_reviews_amazon_cells)
```

Format

A data frame with 1,067 rows and 2 variables

Details

- sentiment. A human scoring of the text.
- text. The sentences from the review.

References

Kotzias, D., Denil, M., De Freitas, N. & Smyth, P. (2015). From group to individual labels using deep features. Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 597-606. <http://mdenil.com/media/papers/2015-deep-multi-instance-learning.pdf>

kotzias_reviews_imdb *Kotzias Reviews: IMBD*

Description

A dataset containing a list of 4 review data sets. Each data set contains sentences with a positive (1) or negative review (-1) taken from reviews of products, movies, & restaurants. The data, compiled by Kotzias, Denil, De Freitas, & Smyth (2015), was originally taken from amazon.com, imdb.com, & yelp.com. Kotzias et al. (2015) provide the following description in the README: "For each website, there exist 500 positive and 500 negative sentences. Those were selected randomly for larger datasets of reviews. We attempted to select sentences that have a clearly positive or negative connotation [sic], the goal was for no neutral sentences to be selected. This data set has been manipulated from the original to be split apart by element (sentence split). The original 0/1 metric has also been converted to -1/1. Please cite Kotzias et al. (2015) if you reuse the data here.

Usage

```
data(kotzias_reviews_imdb)
```

Format

A data frame with 1,041 rows and 2 variables

Details

- sentiment. A human scoring of the text.
- text. The sentences from the review.

References

Kotzias, D., Denil, M., De Freitas, N. & Smyth, P. (2015). From group to individual labels using deep features. Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 597-606. <http://mdenil.com/media/papers/2015-deep-multi-instance-learning.pdf>

kotzias_reviews_yelp *Kotzias Reviews: Yelp*

Description

A dataset containing a list of 4 review data sets. Each data set contains sentences with a positive (1) or negative review (-1) taken from reviews of products, movies, & restaurants. The data, compiled by Kotzias, Denil, De Freitas, & Smyth (2015), was originally taken from amazon.com, imdb.com, & yelp.com. Kotzias et al. (2015) provide the following description in the README: "For each website, there exist 500 positive and 500 negative sentences. Those were selected randomly for larger datasets of reviews. We attempted to select sentences that have a clearly positive or negative connotation [sic], the goal was for no neutral sentences to be selected. This data set has been manipulated from the original to be split apart by element (sentence split). The original 0/1 metric has also been converted to -1/1. Please cite Kotzias et al. (2015) if you reuse the data here.

Usage

```
data(kotzias_reviews_yelp)
```

Format

A data frame with 1,040 rows and 2 variables

Details

- sentiment. A human scoring of the text.
- text. The sentences from the review.

References

Kotzias, D., Denil, M., De Freitas, N. & Smyth, P. (2015). From group to individual labels using deep features. Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 597-606. <http://mdenil.com/media/papers/2015-deep-multi-instance-learning.pdf>

`nyt_articles`*Sentiment Scored New York Times Articles*

Description

A dataset containing Hutto & Gilbert's (2014) sentiment scored New York Times articles.

Usage

```
data(nyt_articles)
```

Format

A data frame with 5,179 rows and 2 variables

Details

- sentiment. A numeric sentiment score
- text. The text from the article

Vadar's License:

The MIT License (MIT)

Copyright (c) 2016 C.J. Hutto

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

References

Hutto, C.J. & Gilbert, E.E. (2014). VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text. Eighth International Conference on Weblogs and Social Media (ICWSM-14). Ann Arbor, MI, June 2014.

<https://github.com/cjhutto/vaderSentiment>

plot.profanity *Plots a profanity object*

Description

Plots a profanity object.

Usage

```
## S3 method for class 'profanity'
plot(x,
      transformation.function = syuzhet::get_dct_transform, ...)
```

Arguments

`x` The profanity object.

`transformation.function`
 A transformation function to smooth the profanity scores.

`...` Other arguments passed to [get_transformed_values](#).

Details

Utilizes Matthew Jocker's **syuzhet** package to calculate smoothed profanity across the duration of the text.

Value

Returns a **ggplot2** object.

plot.profanity_by *Plots a profanity_by object*

Description

Plots a profanity_by object. Red centers are average profanity. Alpha jittered dots are raw sentence level profanity data. Boxes are boxplots.

Usage

```
## S3 method for class 'profanity_by'
plot(x, ordered = TRUE, ...)
```

Arguments

x The profanity_by object.
ordered logical. If TRUE order the output grouping by profanity.
... ignored

Value

Returns a **ggplot2** object.

plot.sentiment	<i>Plots a sentiment object</i>
----------------	---------------------------------

Description

Plots a sentiment object.

Usage

```
## S3 method for class 'sentiment'  
plot(x,  
     transformation.function = syuzhet::get_dct_transform, ...)
```

Arguments

x The sentiment object.
transformation.function A transformation function to smooth the sentiment scores.
... Other arguments passed to [get_transformed_values](#).

Details

Utilizes Matthew Jocker's **syuzhet** package to calculate smoothed sentiment across the duration of the text.

Value

Returns a **ggplot2** object.

`plot.sentiment_by` *Plots a sentiment_by object*

Description

Plots a `sentiment_by` object. Red centers are average sentiment. Alpha jittered dots are raw sentence level sentiment data. Boxes are boxplots.

Usage

```
## S3 method for class 'sentiment_by'  
plot(x, ordered = TRUE, ...)
```

Arguments

<code>x</code>	The <code>sentiment_by</code> object.
<code>ordered</code>	logical. If TRUE order the output grouping by sentiment.
<code>...</code>	ignored

Value

Returns a **ggplot2** object.

`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.extract_profanity_terms
```

Prints an extract_profanity_terms Object

Description

Prints an extract_profanity_terms object

Usage

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

Arguments

x	An extract_profanity_terms object.
...	ignored

```
print.extract_sentiment_terms
```

Prints an extract_sentiment_terms Object

Description

Prints an extract_sentiment_terms object

Usage

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

Arguments

x	An extract_sentiment_terms object.
...	ignored

```
print.validate_sentiment
```

Prints a validate_sentiment Object

Description

Prints a validate_sentiment object

Usage

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

Arguments

x	A validate_sentiment Object
...	ignored.

```
profanity
```

Compute Profanity Rate

Description

Detect the rate of profanity at the sentence level. This method uses a simple dictionary lookup to find profane words and then compute the rate per sentence. The profanity score ranges between 0 (no profanity used) and 1 (all words used were profane). Note that a single profane phrase would count as just one in the profanity_count column but would count as two words in the word_count column.

Usage

```
profanity(text.var, profanity_list = lexicon::profanity_alvarez, ...)
```

Arguments

text.var	The text variable. Can be a get_sentences object or a raw character vector though get_sentences is preferred as it avoids the repeated cost of doing sentence boundary disambiguation every time sentiment is run.
profanity_list	A atomic character vector of profane words. The lexicon package has lists that can be used, including: <ul style="list-style-type: none"> • lexicon::profanity_alvarez • lexicon::profanity_arr_bad • lexicon::profanity_banned • lexicon::profanity_zac_anger • lexicon::profanity_racist
...	ignored.

Value

Returns a **data.table** of:

- `element_id` - The id number of the original vector passed to `profanity`
- `sentence_id` - The id number of the sentences within each `element_id`
- `word_count` - Word count
- `profanity_count` - Count of the number of profane words
- `profanity` - A score of the percentage of profane words

Examples

```
## Not run:
bw <- sample(lexicon::profanity_alvarez, 4)
mytext <- c(
  sprintf('do you like this %s? It is %s. But I hate really bad dogs', bw[1], bw[2]),
  'I am the best friend.',
  NA,
  sprintf('I %s hate this %s', bw[3], bw[4]),
  "Do you really like it? I'm not happy"
)

## works on a character vector but not the preferred method avoiding the
## repeated cost of doing sentence boundary disambiguation every time
## `profanity` is run
profanity(mytext)

## preferred method avoiding paying the cost
mytext2 <- get_sentences(mytext)
profanity(mytext2)

plot(profanity(mytext2))

brady <- get_sentences(crowdflower_deflategate)
brady_swears <- profanity(brady)
brady_swears
hist(brady_swears$profanity)
sum(brady_swears$profanity > 0)
hist(brady_swears$profanity[brady_swears$profanity > 0])

## End(Not run)
```

profanity_by

Profanity Rate By Groups

Description

Approximate the profanity (polarity) of text by grouping variable(s). For a full description of the profanity detection algorithm see [profanity](#). See [profanity](#) for more details about the algorithm, the profanity/valence shifter keys that can be passed into the function, and other arguments that can be passed.

Usage

```
profanity_by(text.var, by = NULL, group.names, ...)
```

Arguments

<code>text.var</code>	The text variable. Also takes a <code>profanityr</code> or <code>profanity_by</code> object.
<code>by</code>	The grouping variable(s). Default <code>NULL</code> uses the original row/element indices; if you used a column of 12 rows for <code>text.var</code> these 12 rows will be used as the grouping variable. Also takes a single grouping variable or a list of 1 or more grouping variables.
<code>group.names</code>	A vector of names that corresponds to <code>group</code> . Generally for internal use.
<code>...</code>	Other arguments passed to profanity .

Value

Returns a **data.table** with grouping variables plus:

- `element_id` - The id number of the original vector passed to `profanity`
- `sentence_id` - The id number of the sentences within each `element_id`
- `word_count` - Word count [summed](#) by grouping variable
- `profanity_count` - The number of profanities used by grouping variable
- `sd` - Standard deviation ([sd](#)) of the sentence level profanity rate by grouping variable
- `ave_profanity` - Profanity rate

Chaining

See the [sentiment_by](#) for details about `sentimentr` chaining.

Examples

```
## Not run:
bw <- sample(lexicon::profanity_alvarez, 4)
mytext <- c(
  sprintf('do you like this %s? It is %s. But I hate really bad dogs', bw[1], bw[2]),
  'I am the best friend.',
  NA,
  sprintf('I %s hate this %s', bw[3], bw[4]),
  "Do you really like it? I'm not happy"
)

## works on a character vector but not the preferred method avoiding the
## repeated cost of doing sentence boundary disambiguation every time
## `profanity` is run
profanity(mytext)
profanity_by(mytext)

## preferred method avoiding paying the cost
mytext <- get_sentences(mytext)
```



```

profanity_by(mytext)
get_sentences(profanity_by(mytext))

(myprofanity <- profanity_by(mytext))
stats::setNames(get_sentences(profanity_by(mytext)),
  round(myprofanity[["ave_profanity"]], 3))

brady <- get_sentences(crowdflower_deflategate)
library(data.table)
bp <- profanity_by(brady)
crowdflower_deflategate[bp[ave_profanity > 0,]$element_id, ]

vulgars <- bp[["ave_profanity"]] > 0
stats::setNames(get_sentences(bp)[vulgars],
  round(bp[["ave_profanity"]][vulgars], 3))

bt <- data.table(crowdflower_deflategate)[,
  source := ifelse(grepl('^RT', text), 'retweet', 'OP')[,
  belichick := grepl('\\bb[A-Za-z]+l[A-Za-z]*ch', text, ignore.case = TRUE)][, ]

prof_bel <- with(bt, profanity_by(text, by = list(source, belichick)))

plot(prof_bel)

## 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.

 sentiment

Polarity Score (Sentiment Analysis)

Description

Approximate the sentiment (polarity) of text by sentence. This function allows the user to easily alter (add, change, replace) the default polarity and valence shifters dictionaries to suit the context dependent needs of a particular data set. See the `polarity_dt` and `valence_shifters_dt` arguments for more information. Other hyper-parameters may add additional fine tuned control of the algorithm that may boost performance in different contexts.

Usage

```
sentiment(text.var, polarity_dt = lexicon::hash_sentiment_jockers_rinker,
  valence_shifters_dt = lexicon::hash_valence_shifters, hyphen = "",
  amplifier.weight = 0.8, n.before = 5, n.after = 2,
  question.weight = 1, adversative.weight = 0.25,
  neutral.nonverb.like = FALSE, missing_value = 0, ...)
```

Arguments

`text.var` The text variable. Can be a `get_sentences` object or a raw character vector though `get_sentences` is preferred as it avoids the repeated cost of doing sentence boundary disambiguation every time `sentiment` is run.

`polarity_dt` A **data.table** of positive/negative words and weights with `x` and `y` as column names. The **lexicon** package has several dictionaries that can be used, including:

- `lexicon::hash_sentiment_jockers_rinker`
- `lexicon::hash_sentiment_jockers`
- `lexicon::emojis_sentiment`
- `lexicon::hash_sentiment_emojis`
- `lexicon::hash_sentiment_huliu`
- `lexicon::hash_sentiment_loughran_mcdonald`
- `lexicon::hash_sentiment_nrc`
- `lexicon::hash_sentiment_sentinet`
- `lexicon::hash_sentiment_sentiword`
- `lexicon::hash_sentiment_slagsd`
- `lexicon::hash_sentiment_socal_google`

Additionally, the `as_key` function can be used to make a sentiment frame suitable for `polarity_dt`. This takes a 2 column data.frame with the first column being words and the second column being polarity values. Note that as of version 1.0.0 **sentimentr** switched from the Liu & HU (2004) dictionary as the default to Jocker's (2017) dictionary from the **syuzhet** package. Use `lexicon::hash_sentiment_huliu` to obtain the old behavior.

valence_shifters_dt	A data.table of valence shifters that can alter a polarized word's meaning and an integer key for negators (1), amplifiers [intensifiers] (2), de-amplifiers [downtoners] (3) and adversative conjunctions (4) with x and y as column names.
hyphen	The character string to replace hyphens with. Default replaces with nothing so 'sugar-free' becomes 'sugarfree'. Setting hyphen = " " would result in a space between words (e.g., 'sugar free'). Typically use either " " or default "".
amplifier.weight	The weight to apply to amplifiers/de-amplifiers [intensifiers/downtoners] (values from 0 to 1). This value will multiply the polarized terms by 1 + this value.
n.before	The number of words to consider as valence shifters before the polarized word. To consider the entire beginning portion of a sentence use n.before = Inf.
n.after	The number of words to consider as valence shifters after the polarized word. To consider the entire ending portion of a sentence use n.after = Inf.
question.weight	The weighting of questions (values from 0 to 1). Default is 1. A 0 corresponds with the belief that questions (pure questions) are not polarized. A weight may be applied based on the evidence that the questions function with polarized sentiment. In an opinion tasks such as a course evaluation the questions are more likely polarized, not designed to gain information. On the other hand, in a setting with more natural dialogue, the question is less likely polarized and is likely to function as a means to gather information.
adversative.weight	The weight to give to adversative conjunctions or contrasting conjunctions (e.g., "but") that overrule the previous clause (Halliday & Hasan, 2013). Weighting a contrasting statement stems from the belief that the adversative conjunctions like "but", "however", and "although" amplify the current clause and/or down weight the prior clause. If an adversative conjunction is located before the polarized word in the context cluster the cluster is up-weighted $1 + \text{number of occurrences of the adversative conjunctions before the polarized word} \times \text{weight given}$ ($1 + N_{\text{adversative conjunctions}} * z_2$ where z_2 is the adversative.weight). Conversely, an adversative conjunction found after the polarized word in a context cluster down weights the cluster $1 - \text{number of occurrences of the adversative conjunctions after the polarized word} \times \text{weight given}$ ($1 + N_{\text{adversative conjunctions}} * -1 * z_2$). These are added to the deamplifier and amplifier weights and thus the down weight is constrained to -1 as the lower bound. Set to zero to remove adversative conjunction weighting.
neutral.nonverb.like	logical. If TRUE, and 'like' is found in the polarity_dt, when the word 'like' is preceded by one of the following linking verbs: "'s", "was", "is", "has", "am", "are", "'re", "had", or "been" it is neutralized as this non-verb form of like is not likely polarized. This is a poor man's part of speech tagger, maintaining the balance between speed and accuracy. The word 'like', as a verb, tends to be polarized and is usually preceded by a noun or pronoun, not one of the linking verbs above. This hyper parameter doesn't always yield improved results depending on the context of where the text data comes from. For example, it is likely to be more useful in literary works, where like is often used in non-verb

form, than product comments. Use of this parameter will add compute time, this must be weighed against the need for accuracy and the likelihood that more accurate results will come from setting this argument to TRUE.

missing_value A value to replace NA/NaN with. Use NULL to retain missing values.
 ... Ignored.

Details

The equation used by the algorithm to assign value to polarity of each sentence first utilizes the sentiment dictionary to tag polarized words. Each paragraph ($p_i = \{s_1, s_2, \dots, s_n\}$) composed of sentences, is broken into element sentences ($s_i, j = \{w_1, w_2, \dots, w_n\}$) where w are the words within sentences. Each sentence (s_j) is broken into an ordered bag of words. Punctuation is removed with the exception of pause punctuations (commas, colons, semicolons) which are considered a word within the sentence. I will denote pause words as cw (comma words) for convenience. We can represent these words as an i,j,k notation as $w_{i,j,k}$. For example $w_{3,2,5}$ would be the fifth word of the second sentence of the third paragraph. While I use the term paragraph this merely represents a complete turn of talk. For example t may be a cell level response in a questionnaire composed of sentences.

The words in each sentence ($w_{i,j,k}$) are searched and compared to a dictionary of polarized words (e.g., Jockers (2017) dictionary found in the **lexicon** package). Positive ($w_{i,j,k}^+$) and negative ($w_{i,j,k}^-$) words are tagged with a +1 and -1 respectively. I will denote polarized words as pw for convenience. These will form a polar cluster ($c_{i,j,l}$) which is a subset of the sentence ($c_{i,j,l} \subseteq s_i, j$).

The polarized context cluster ($c_{i,j,l}$) of words is pulled from around the polarized word (pw) and defaults to 4 words before and two words after pw to be considered as valence shifters. The cluster can be represented as ($c_{i,j,l} = \{pw_{i,j,k-nb}, \dots, pw_{i,j,k}, \dots, pw_{i,j,k+na}\}$), where nb & na are the parameters `n.before` and `n.after` set by the user. The words in this polarized context cluster are tagged as neutral ($w_{i,j,k}^0$), negator ($w_{i,j,k}^n$), amplifier [intensifier] ($w_{i,j,k}^a$), or de-amplifier [downtoner] ($w_{i,j,k}^d$). Neutral words hold no value in the equation but do affect word count (n). Each polarized word is then weighted (w) based on the weights from the `polarity_dt` argument and then further weighted by the function and number of the valence shifters directly surrounding the positive or negative word (pw). Pause (cw) locations (punctuation that denotes a pause including commas, colons, and semicolons) are indexed and considered in calculating the upper and lower bounds in the polarized context cluster. This is because these marks indicate a change in thought and words prior are not necessarily connected with words after these punctuation marks. The lower bound of the polarized context cluster is constrained to $\max\{pw_{i,j,k-nb}, 1, \max\{cw_{i,j,k} < pw_{i,j,k}\}\}$ and the upper bound is constrained to $\min\{pw_{i,j,k+na}, w_{i,j,n}, \min\{cw_{i,j,k} > pw_{i,j,k}\}\}$ where $w_{i,j,n}$ is the number of words in the sentence.

The core value in the cluster, the polarized word is acted upon by valence shifters. Amplifiers (intensifiers) increase the polarity by 1.8 (.8 is the default weight (z)). Amplifiers ($w_{i,j,k}^a$) become de-amplifiers if the context cluster contains an odd number of negators ($w_{i,j,k}^n$). De-amplifiers (downtoners) work to decrease the polarity. Negation ($w_{i,j,k}^n$) acts on amplifiers/de-amplifiers as discussed but also flip the sign of the polarized word. Negation is determined by raising -1 to the power of the number of negators ($w_{i,j,k}^n + 2$). Simply, this is a result of a belief that two negatives equal a positive, 3 negatives a negative and so on.

The adversative conjunctions (i.e., 'but', 'however', and 'although') also weight the context cluster. An adversative conjunction before the polarized word ($w_{adversative\ conjunction}, \dots, w_{i,j,k}^p$) up-weights the cluster by $1 + z_2 * \{|w_{adversative\ conjunction}|, \dots, w_{i,j,k}^p\}$ (.85 is the default weight (z_2)).

An adversative conjunction after the polarized word down-weights the cluster by $1 + \{w_{i,j,k}^p, \dots, |w_{adversative\ conjunction}|\} * z_2$. The number of occurrences before and after the polarized word are multiplied by 1 and -1 respectively and then summed within context cluster. It is this value that is multiplied by the weight and added to 1. This corresponds to the belief that an adversative conjunction makes the next clause of greater values while lowering the value placed on the prior clause.

The researcher may provide a weight z to be utilized with amplifiers/de-amplifiers (default is .8; de-amplifier weight is constrained to -1 lower bound). Last, these weighted context clusters ($c_{i,j,l}$) are summed ($c'_{i,j}$) and divided by the square root of the word count ($\sqrt{w_{i,j,n}}$) yielding an **unbounded polarity score** (δ) for each sentence.

$$\delta = \frac{c'_{i,j}}{\sqrt{w_{i,j,n}}}$$

Where:

$$c'_{i,j} = \sum ((1 + w_{amp} + w_{deamp}) \cdot w_{i,j,k}^p (-1)^{2+w_{neg}})$$

$$w_{amp} = (w_b > 1) + \sum (w_{neg} \cdot (z \cdot w_{i,j,k}^a))$$

$$w_{deamp} = \max(w_{deamp'}, -1)$$

$$w_{deamp'} = (w_b < 1) + \sum (z(-w_{neg} \cdot w_{i,j,k}^a + w_{i,j,k}^d))$$

$$w_b = 1 + z_2 * w_{b'}$$

$$w_{b'} = \sum (|w_{adversative\ conjunction}|, \dots, w_{i,j,k}^p, w_{i,j,k}^d, \dots, |w_{adversative\ conjunction}| * -1)$$

$$w_{neg} = \left(\sum w_{i,j,k}^n \right) \bmod 2$$

Value

Returns a **data.table** of:

- element_id - The id number of the original vector passed to sentiment
- sentence_id - The id number of the sentences within each element_id
- word_count - Word count
- sentiment - Sentiment/polarity score

Note

The polarity score is dependent upon the polarity dictionary used. This function defaults to a combined and augmented version of Jocker's (2017) [originally exported by the **syuzhet** package] & Rinker's augmented Hu & Liu (2004) dictionaries in the **lexicon** package, however, this may not be appropriate, for example, in the context of children in a classroom. The user may (is encouraged) to provide/augment the dictionary (see the `as_key` function). For instance the word "sick" in a high school setting may mean that something is good, whereas "sick" used by a typical adult indicates something is not right or negative connotation (**deixis**).

References

- Jockers, M. L. (2017). Syuzhet: Extract sentiment and plot arcs from text. Retrieved from <https://github.com/mjockers/syuzhet>
- Hu, M., & Liu, B. (2004). Mining opinion features in customer reviews. National Conference on Artificial Intelligence.
- Halliday, M. A. K. & Hasan, R. (2013). Cohesion in English. New York, NY: Routledge.
<http://www.slideshare.net/jeffreymbreen/r-by-example-mining-twitter-for>
<http://hedonometer.org/papers.html> Links to papers on hedonometrics

See Also

<https://github.com/trestletech/Sermon-Sentiment-Analysis>

Other sentiment functions: [sentiment_by](#)

Examples

```
mytext <- c(
  'do you like it? But I hate really bad dogs',
  'I am the best friend.',
  "Do you really like it? I'm not a fan",
  "It's like a tree."
)

## works on a character vector but not the preferred method avoiding the
## repeated cost of doing sentence boundary disambiguation every time
## `sentiment` is run. For small batches the loss is minimal.
## Not run:
sentiment(mytext)

## End(Not run)

## preferred method avoiding paying the cost
mytext <- get_sentences(mytext)
sentiment(mytext)
sentiment(mytext, question.weight = 0)

sam_dat <- get_sentences(gsub("Sam-I-am", "Sam I am", sam_i_am))
(sam <- sentiment(sam_dat))
plot(sam)
plot(sam, scale_range = TRUE, low_pass_size = 5)
```

```

plot(sam, scale_range = TRUE, low_pass_size = 10)

plot(sam, transformation.function = syuzhet::get_transformed_values)
plot(sam, transformation.function = syuzhet::get_transformed_values,
      scale_range = TRUE, low_pass_size = 5)

y <- get_sentences(
  "He was not the sort of man that one would describe as especially handsome."
)
sentiment(y)
sentiment(y, n.before=Inf)

dat <- data.frame(
  w = c('Person 1', 'Person 2'),
  x = c(paste0(
    "Mr. Brown is nasty! He says hello. i give him rage. i will ",
    "go at 5 p. m. eastern time. Angry thought in between!go there"
  ), "One more thought for the road! I am going now. Good day and good riddance."),
  y = state.name[c(32, 38)],
  z = c(.456, .124),
  stringsAsFactors = FALSE
)
sentiment(get_sentences(dat$x))
sentiment(get_sentences(dat))

## Not run:
## tidy approach
library(dplyr)
library(magrittr)

hu_liu_cannon_reviews %>%
  mutate(review_split = get_sentences(text)) %$%
  sentiment(review_split)

## End(Not run)

## Emojis
## Not run:
## Load R twitter data
x <- read.delim(system.file("docs/r_tweets.txt", package = "textclean"),
  stringsAsFactors = FALSE)

x

library(dplyr); library(magrittr)

## There are 2 approaches
## Approach 1: Replace with words
x %>%
  mutate(Tweet = replace_emoji(Tweet)) %$%
  sentiment(Tweet)

## Approach 2: Replace with identifier token

```

```

combined_emoji <- update_polarity_table(
  lexicon::hash_sentiment_jockers_rinker,
  x = lexicon::hash_sentiment_emojis
)

x %>%
  mutate(Tweet = replace_emoji_identifier(Tweet)) %>%
  sentiment(Tweet, polarity_dt = combined_emoji)

## End(Not run)

```

sentimentr	<i>Calculate Text Polarity Sentiment</i>
------------	--

Description

Calculate text polarity sentiment at the sentence level and optionally aggregate by rows or grouping variable(s).

sentiment_attributes	<i>Extract Sentiment Attributes from Text</i>
----------------------	---

Description

This function utilizes **gofastr** and **termco** to extract sentiment based attributes (attributes concerning polarized words and valence shifters) from a text. Attributes include the rate of polarized terms and valence shifters relative to number of words. Additionally, cooccurrence rates for valence shifters are computed.

Usage

```

sentiment_attributes(text.var,
  polarity_dt = lexicon::hash_sentiment_jockers_rinker,
  valence_shifters_dt = lexicon::hash_valence_shifters, ...)

```

Arguments

text.var	The text variable.
polarity_dt	A data.table of positive/negative words and weights with x and y as column names.
valence_shifters_dt	A data.table of valence shifters that can alter a polarized word's meaning and an integer key for negators (1), amplifiers(2), de-amplifiers (3) and adversative conjunctions (4) with x and y as column names.
...	ignored.

Value

Returns a list of four items:

Meta	The number of words, sentences, and questions in the text
Attributes	The rate of sentiment attributes relative to the number of words
Polarized_Cooccurrences	The rate that valence shifters cooccur with a polarized word in the same sentence
Cooccurrences	A cooccurrence matrix of sentiment attributes; ‘polarized’ is the sum of positive and negative

Note

gofastr and **termco** must be installed. If they are not (which they are not part of **sentimentr** install) then the function will prompt you to attempt to install them using `install.packages` and `ghit::install_github`.

Examples

```
## Not run:
sentiment_attributes(presidential_debates_2012$dialogue)

## End(Not run)
```

sentiment_by

Polarity Score (Sentiment Analysis) By Groups

Description

Approximate the sentiment (polarity) of text by grouping variable(s). For a full description of the sentiment detection algorithm see [sentiment](#). See [sentiment](#) for more details about the algorithm, the sentiment/valence shifter keys that can be passed into the function, and other arguments that can be passed.

Usage

```
sentiment_by(text.var, by = NULL,
             averaging.function = sentimentr::average_downweighted_zero,
             group.names, ...)
```

Arguments

<code>text.var</code>	The text variable. Also takes a <code>sentimentr</code> or <code>sentiment_by</code> object.
<code>by</code>	The grouping variable(s). Default <code>NULL</code> uses the original row/element indices; if you used a column of 12 rows for <code>text.var</code> these 12 rows will be used as the grouping variable. Also takes a single grouping variable or a list of 1 or more grouping variables.

averaging.function

A function for performing the group by averaging. The default, `average_downweighted_zero`, downweights zero values in the averaging. Note that the function must handle NAs. The **sentimentr** functions `average_weighted_mixed_sentiment` and `average_mean` are also available. The former upweights negative when the analysts suspects the speaker is likely to surround negatives with positives (mixed) as a polite social convention but still the affective state is negative. The later is a standard mean average.

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

`...` Other arguments passed to `sentiment`.

Value

Returns a **data.table** with grouping variables plus:

- `element_id` - The id number of the original vector passed to `sentiment`
- `sentence_id` - The id number of the sentences within each `element_id`
- `word_count` - Word count summed by grouping variable
- `sd` - Standard deviation (`sd`) of the sentiment/polarity score by grouping variable
- `ave_sentiment` - Sentiment/polarity score mean average by grouping variable

Chaining

sentimentr uses non-standard evaluation when you use `with()` OR `%%` (**magrittr**) and looks for the vectors within the data set passed to it. There is one exception to this...when you pass a `get_sentences()` object to `sentiment_by()` to the first argument which is `text.var` it calls the `sentiment_by.get_sentences_data_frame` method which requires `text.var` to be a `get_sentences_data_frame` object. Because this object is a `data.frame` its method knows this and knows it can access the columns of the `get_sentences_data_frame` object directly (usually `text.var` is an atomic vector), it just needs the names of the columns to grab.

To illustrate this point understand that all three of these approaches result in exactly the same output:

```
## method 1
presidential_debates_2012 %>%
  get_sentences() %>%
  sentiment_by(by = c('person', 'time'))

## method 2
presidential_debates_2012 %>%
  get_sentences() %%%
  sentiment_by(., by = c('person', 'time'))

## method 3
presidential_debates_2012 %>%
  get_sentences() %%%
  sentiment_by(dialogue, by = list(person, time))
```

Also realize that a `get_sentences_data_frame` object also has a column with a `get_sentences_character` class column which also has a method in **sentimentr**.

When you use `with()` OR `%%` then you're not actually passing the `get_sentences_data_frame` object to **sentimentr** and hence the `sentiment_by.get_sentences_data_frame` method isn't called rather `sentiment_by` is evaluated in the environment/data of the `get_sentences_data_frame` object. You can force the object passed this way to be evaluated as a `get_sentences_data_frame` object and thus calling the `sentiment_by.get_sentences_data_frame` method by using the `.` operator as I've done in method 2 above. Otherwise you pass the name of the text column which is actually a `get_sentences_character` class and it calls its own method. In this case the `by` argument expects vectors or a list of vectors and since it's being evaluated within the data set you can use `list()`.

See Also

Other sentiment functions: [sentiment](#)

Examples

```
mytext <- c(
  'do you like it? It is red. But I hate really bad dogs',
  'I am the best friend.',
  "Do you really like it? I'm not happy"
)

## works on a character vector but not the preferred method avoiding the
## repeated cost of doing sentence boundary disambiguation every time
## `sentiment` is run
## Not run:
sentiment(mytext)
sentiment_by(mytext)

## End(Not run)

## preferred method avoiding paying the cost
mytext <- get_sentences(mytext)

sentiment_by(mytext)
sentiment_by(mytext, averaging.function = average_mean)
sentiment_by(mytext, averaging.function = average_weighted_mixed_sentiment)
get_sentences(sentiment_by(mytext))

(mysentiment <- sentiment_by(mytext, question.weight = 0))
stats::setNames(get_sentences(sentiment_by(mytext, question.weight = 0)),
  round(mysentiment[["ave_sentiment"]], 3))

pres_dat <- get_sentences(presidential_debates_2012)

## Not run:
## less optimized way
with(presidential_debates_2012, sentiment_by(dialogue, person))
```

```
## End(Not run)

## Not run:
sentiment_by(pres_dat, 'person')

(out <- sentiment_by(pres_dat, c('person', 'time')))
plot(out)
plot(uncombine(out))

sentiment_by(out, presidential_debates_2012$person)
with(presidential_debates_2012, sentiment_by(out, time))

highlight(with(presidential_debates_2012, sentiment_by(out, list(person, time))))

## End(Not run)

## Not run:
## tidy approach
library(dplyr)
library(magrittr)

hu_liu_cannon_reviews %>%
  mutate(review_split = get_sentences(text)) %>%
  sentiment_by(review_split)

## End(Not run)
```

uncombine

Ungroup a sentiment_by Object to the Sentence Level

Description

Ungroup a sentiment_by object, stretching to the element_id and sentence_id levels.

Usage

```
uncombine(x, ...)
```

Arguments

x	A sentiment_by object.
...	Ignored.

Value

Returns a **data.table** with grouping variables plus:

- element_id - The id number of the original vector passed to sentiment
- word_count - Word count **summed** by grouping variable
- sd - Standard deviation (**sd**) of the sentiment/polarity score by grouping variable
- ave_sentiment - Sentiment/polarity score **mean** average by grouping variable

Examples

```
mytext <- c(
  'do you like it? But I hate really bad dogs',
  'I am the best friend.',
  "Do you really like it? I'm not happy"
)

mytext <- get_sentences(mytext)
(x <- sentiment_by(mytext))
uncombine(x)

(y <- with(
  presidential_debates_2012,
  sentiment_by(
    text.var = get_sentences(dialogue),
    by = list(person, time)
  )
))
uncombine(y)
```

validate_sentiment *Validate Sentiment Score Sign Against Known Results*

Description

Provides a multiclass macroaverage/microaverage of precision, recall, accuracy, and F-score for the sign of the predicted sentiment against known sentiment scores. There are three classes sentiment analysis generally predicts: positive (> 0), negative (< 0) and neutral ($= 0$). In assessing model performance one can use macro- or micro- averaging across classes. Macroaveraging allows every class to have an equal say. Microaveraging gives larger say to larger classes.

Usage

```
validate_sentiment(predicted, actual, ...)
```

Arguments

predicted	A numeric vector of predicted sentiment scores or a sentimentr object that returns sentiment scores.
actual	A numeric vector of known sentiment ratings.
...	ignored.

Value

Returns a [data.frame](#) with a macroaveraged and microaveraged model validation scores. Additionally, the [data.frame](#) has the following attributes:

confusion_matrix	A confusion matrix of all classes
------------------	-----------------------------------

```

class_confusion_matrices  A list of class level (class vs. all) confusion matrices
macro_stats               A data.frame of the macroaveraged class level stats before averaging
mda                       Mean Directional Accuracy
mare                      Mean Absolute Rescaled Error

```

Note

Mean Absolute Rescaled Error (MARE) is defined as: $\sum \frac{|actual - predicted|}{2n}$ and gives a sense of, on average, how far off were the rescaled predicted values (-1 to 1) from the rescaled actual values (-1 to 1). A value of 0 means perfect accuracy. A value of 1 means perfectly wrong every time. A value of .5 represents expected value for random guessing. This measure is related to **Mean Absolute Error**.

References

<https://www.youtube.com/watch?v=0wwdYHWRB5E&index=31&list=PL6397E4B26D00A269>
[https://en.wikipedia.org/wiki/Mean_Directional_Accuracy_\(MDA\)](https://en.wikipedia.org/wiki/Mean_Directional_Accuracy_(MDA))

Examples

```

actual <- c(1, 1, 1, 1, -1, -1, -1, -1, -1, -1, -1, 1,-1)
predicted <- c(1, 0, 1, -1, 1, 0, -1, -1, -1, -1, 0, 1,-1)
validate_sentiment(predicted, actual)

scores <- hu_liu_cannon_reviews$sentiment
mod <- sentiment_by(get_sentences(hu_liu_cannon_reviews$text))

validate_sentiment(mod$ave_sentiment, scores)
validate_sentiment(mod, scores)

x <- validate_sentiment(mod, scores)
attributes(x)$confusion_matrix
attributes(x)$class_confusion_matrices
attributes(x)$macro_stats

## Annie Swafford Example
swafford <- data.frame(
  text = c(
    "I haven't been sad in a long time.",
    "I am extremely happy today.",
    "It's a good day.",
    "But suddenly I'm only a little bit happy.",
    "Then I'm not happy at all.",
    "In fact, I am now the least happy person on the planet.",
    "There is no happiness left in me.",
    "Wait, it's returned!",
    "I don't feel so bad after all!"
  ),
  actual = c(.8, 1, .8, -.1, -.5, -1, -1, .5, .6),
  stringsAsFactors = FALSE

```

```
)  
  
pred <- sentiment_by(swafford$text)  
validate_sentiment(  
  pred,  
  actual = swafford$actual  
)
```

Index

- *Topic **bad-words**
 - profanity, 30
 - profanity_by, 31
- *Topic **cursing**,
 - profanity, 30
 - profanity_by, 31
- *Topic **cussing**,
 - profanity, 30
 - profanity_by, 31
- *Topic **datasets**
 - course_evaluations, 9
 - crowdflower_deflategate, 9
 - crowdflower_products, 10
 - crowdflower_self_driving_cars, 11
 - crowdflower_weather, 11
 - hotel_reviews, 17
 - hu_liu_apex_reviews, 18
 - hu_liu_cannon_reviews, 19
 - hu_liu_jukebox_reviews, 19
 - hu_liu_nikon_reviews, 20
 - hu_liu_nokia_reviews, 21
 - kaggle_movie_reviews, 22
 - kotzias_reviews_amazon_cells, 22
 - kotzias_reviews_imdb, 23
 - kotzias_reviews_yelp, 24
 - nyt_articles, 25
 - presidential_debates_2012, 28
 - sam_i_am, 33
- *Topic **group**
 - sentiment_by, 41
- *Topic **hash**
 - as_key, 2
- *Topic **key**
 - as_key, 2
- *Topic **lookup**
 - as_key, 2
- *Topic **polarity**,
 - sentiment_by, 41
- *Topic **polarity**
 - sentiment, 34
- *Topic **profanity**,
 - profanity, 30
 - profanity_by, 31
- *Topic **sentiment**,
 - sentiment, 34
 - sentiment_by, 41
- *Topic **vulgarity**,
 - profanity, 30
 - profanity_by, 31
- as_key, 2
- available_data, 5
- average_downweighted_zero, 7, 42
- average_mean
 - (average_downweighted_zero), 7
- average_weighted_mixed_sentiment
 - (average_downweighted_zero), 7
- combine_data, 8
- course_evaluations, 9
- crowdflower_deflategate, 9
- crowdflower_products, 10
- crowdflower_self_driving_cars, 11
- crowdflower_weather, 11
- data.frame, 3, 45, 46
- extract_profanity_terms, 12
- extract_sentiment_terms, 13
- general_rescale, 14
- get_sentences, 15, 16
- get_sentences2, 16
- get_transformed_values, 26, 27
- highlight, 16
- hotel_reviews, 17
- hu_liu_apex_reviews, 18
- hu_liu_cannon_reviews, 19
- hu_liu_jukebox_reviews, 19

hu_liu_nikon_reviews, 20
hu_liu_nokia_reviews, 21

is_key (as_key), 2

kaggle_movie_reviews, 22
kotzias_reviews_amazon_cells, 22
kotzias_reviews_imdb, 23
kotzias_reviews_yelp, 24

list, 46

mean, 42, 44

nyt_articles, 25

package-sentiment (sentimentr), 40
plot.profanity, 26
plot.profanity_by, 26
plot.sentiment, 27
plot.sentiment_by, 28
presidential_debates_2012, 28
print.extract_profanity_terms, 29
print.extract_sentiment_terms, 29
print.validate_sentiment, 30
profanity, 30, 31, 32
profanity_by, 31

sam_i_am, 33
sd, 32, 42, 44
sentiment, 3, 34, 41–43
sentiment_attributes, 40
sentiment_by, 32, 38, 41
sentimentr, 40
sentimentr-package (sentimentr), 40
sentimentr_data (available_data), 6
split_sentence, 15
sum, 32, 42, 44

uncombine, 44
update_key (as_key), 2
update_polarity_table (as_key), 2
update_valence_shifter_table (as_key), 2

validate_sentiment, 45